



IntechOpen

Frontiers in Evolutionary Robotics

Edited by Hitoshi Iba



FRONTIERS IN EVOLUTIONARY ROBOTICS

EDITED BY
HITOSHI IBA

Frontiers in Evolutionary Robotics

<http://dx.doi.org/10.5772/62>

Edited by Hitoshi Iba

© The Editor(s) and the Author(s) 2008

The moral rights of the and the author(s) have been asserted.

All rights to the book as a whole are reserved by INTECH. The book as a whole (compilation) cannot be reproduced, distributed or used for commercial or non-commercial purposes without INTECH's written permission.

Enquiries concerning the use of the book should be directed to INTECH rights and permissions department (permissions@intechopen.com).

Violations are liable to prosecution under the governing Copyright Law.



Individual chapters of this publication are distributed under the terms of the Creative Commons Attribution 3.0 Unported License which permits commercial use, distribution and reproduction of the individual chapters, provided the original author(s) and source publication are appropriately acknowledged. If so indicated, certain images may not be included under the Creative Commons license. In such cases users will need to obtain permission from the license holder to reproduce the material. More details and guidelines concerning content reuse and adaptation can be found at <http://www.intechopen.com/copyright-policy.html>.

Notice

Statements and opinions expressed in the chapters are those of the individual contributors and not necessarily those of the editors or publisher. No responsibility is accepted for the accuracy of information contained in the published chapters. The publisher assumes no responsibility for any damage or injury to persons or property arising out of the use of any materials, instructions, methods or ideas contained in the book.

First published in Croatia, 2008 by INTECH d.o.o.

eBook (PDF) Published by IN TECH d.o.o.

Place and year of publication of eBook (PDF): Rijeka, 2019.

IntechOpen is the global imprint of IN TECH d.o.o.

Printed in Croatia

Legal deposit, Croatia: National and University Library in Zagreb

Additional hard and PDF copies can be obtained from orders@intechopen.com

Frontiers in Evolutionary Robotics

Edited by Hitoshi Iba

p. cm.

ISBN 978-3-902613-19-6

eBook (PDF) ISBN 978-953-51-5829-5

We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,000+

Open access books available

116,000+

International authors and editors

120M+

Downloads

151

Countries delivered to

Our authors are among the
Top 1%

most cited scientists

12.2%

Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



Preface

This book presented techniques and experimental results which have been pursued for the purpose of evolutionary robotics. Evolutionary robotics is a new method for the automatic creation of autonomous robots. When executing tasks by autonomous robots, we can make the robot learn what to do so as to complete the task from interactions with its environment, but not manually pre-program for all situations. Many researchers have been studying the techniques for evolutionary robotics by using Evolutionary Computation (EC), such as Genetic Algorithms (GA) or Genetic Programming (GP). Their goal is to clarify the applicability of the evolutionary approach to the real-robot learning, especially, in view of the adaptive robot behavior as well as the robustness to noisy and dynamic environments. For this purpose, authors in this book explain a variety of real robots in different fields.

For instance, in a multi-robot system, several robots simultaneously work to achieve a common goal via interaction; their behaviors can only emerge as a result of evolution and interaction. How to learn such behaviors is a central issue of Distributed Artificial Intelligence (DAI), which has recently attracted much attention. This book addresses the issue in the context of a multi-robot system, in which multiple robots are evolved using EC to solve a cooperative task. Since directly using EC to generate a program of complex behaviors is often very difficult, a number of extensions to basic EC are proposed in this book so as to solve these control problems of the robot.

Another important topic is a humanoid robot. In recent years, many researches have been conducted upon various aspects of humanoid robots. Since humanoid robots have physical features similar to us, it is very important to let them behave intelligently like humans. In addition, from the viewpoint of AI or DAI, it is rewarding to study how cooperatively humanoid robots perform a task just as we humans can. However, there have been very few studies on the cooperative behaviors of multiple humanoid robots. Thus, in this book, several authors describe the emergence of the intelligent humanoid behavior through the usage of EC.

There are many other interesting research topics described in this book, such as evolutionary learning of robot parameters, evolutionary design, evolutionary motion control, evolutionary morphology and evolutionary navigation planning.

The developments in this book aim to facilitate the discovery of evolutionary framework in the sense that the empirical studies are mainly from the fields of real-world robotics domains, e.g., Khepera, AIBO and humanoid robots. In other words, the principles of evolutionary computation can be effectively applied to many real-world tasks. The presented successful experimental results suggest that, in general, the proposed methodologies can be useful in realizing evolutionary robotics.

Editor

Hitoshi Iba

University of Tokyo
Japan

Contents

Preface	VII
1. A Comparative Evaluation of Methods for Evolving a Cooperative Team Takaya Arita and Yasuyuki Suzuki	001
2. An Adaptive Penalty Method for Genetic Algorithms in Constrained Optimization Problems Helio J. C. Barbosa and Afonso C. C. Lemonge	009
3. Evolutionary-Based Control Approaches for Multirobot Systems Jekanthan Thangavelautham, Timothy D. Barfoot and Gabriele M.T. D'Eleuterio	035
4. Learning by Experience and by Imitation in Multi-Robot Systems Dennis Barrios-Aranibar, Luiz M. G. Gonçalves and Pablo Javier Alsina	061
5. Cellular Non-linear Networks as a New Paradigm for Evolutionary Robotics Eleonora Bilotta and Pietro Pantano	087
6. Optimal Design of Mechanisms for Robot Hands J.A. Cabrera, F. Nadal and A. Simón	109
7. Evolving Humanoids: Using Artificial Evolution as an Aid in the Design of Humanoid Robots Malachy Eaton	127
8. Real-Time Evolutionary Algorithms for Constrained Predictive Control Mario Luca Fravolini, Antonio Ficola and Michele La Cava	139
9. Applying Real-Time Survivability Considerations in Evolutionary Behavior Learning by a Mobile Robot Wolfgang Freund, Tomás Arredondo V. and César Munoz	185
10. An Evolutionary MAP Filter for Mobile Robot Global Localization L. Moreno, S. Garrido, M. L. Muñoz and D. Blanco	197
11. Learning to Walk with Model Assisted Evolution Strategies Matthias Hebbel and Walter Nisticò	209
12. Evolutionary Morphology for Polycube Robots Takahiro Tohge and Hitoshi Iba	221

13. Mechanism of Emergent Symmetry Properties on Evolutionary Robotic System	233
Naohide Yasuda, Takuma Kawakami, Hiroaki Iwano, Katsuya Kanai, Koki Kikuchi and Xueshan Gao	
14. A Quantitative Analysis of Memory Usage for Agent Tasks	247
DaeEun Kim	
15. Evolutionary Parametric Identification of Dynamic Systems	275
Dimitris Koulocheris and Vasilis Dertimanis	
16. Evolutionary Computation of Multi-robot/agent Systems	291
Philippe Lucidarme	
17. Embodiment of Legged Robots Emerged in Evolutionary Design: Pseudo Passive Dynamic Walkers	311
Kojiro Matsushita and Hiroshi Yokoi	
18. Action Selection and Obstacle Avoidance using Ultrasonic and Infrared Sensors	327
Fernando Montes-González, Daniel Flandes-Eusebio and Luis Pellegrin-Zazueta	
19. Multi-Legged Robot Control Using GA-Based Q-Learning Method With Neighboring Crossover	341
Tadahiko Murata and Masatoshi Yamaguchi	
20. Evolved Navigation Control for Unmanned Aerial Vehicles	353
Gregory J. Barlow and Choong K. Oh	
21. Application of Artificial Evolution to Obstacle Detection and Mobile Robot Control	379
Olivier Pauplin and Arnaud de La Fortelle	
22. Hunting in an Environment Containing Obstacles: A Combinatory Study of Incremental Evolution and Co-evolutionary Approaches	393
Ioannis Mermigkis and Loukas Petrou	
23. Evolving Behavior Coordination for Mobile Robots using Distributed Finite-State Automata	413
Pavel Petrovic	
24. An Embedded Evolutionary Controller to Navigate a Population of Autonomous Robots	439
Eduardo do Valle Simões	
25. Optimization of a 2 DOF Micro Parallel Robot Using Genetic Algorithms	465
Sergiu-Dan Stan, Vistrian Maties and Radu Balan	

26. Progressive Design through Staged Evolution Ricardo A. Téllez and Cécilio Angulo	491
27. Emotional Intervention on Stigmergy Based Foraging Behaviour of Immune Network Driven Mobile Robots Diana Tsankova	517
28. Evolutionary Distributed Control of a Biologically Inspired Modular Robot Sunil Pranit Lal and Koji Yamada	543
29. Evolutionary Motion Design for Humanoid Robots Toshihiko Yanase and Hitoshi Iba	567

A Comparative Evaluation of Methods for Evolving a Cooperative Team

Takaya Arita¹ and Yasuyuki Suzuki²

¹*Nagoya University*, ²*Goldman Sachs Japan Holdings*
Japan

1. Introduction

Some problems can be efficiently solved only by teams consisting of cooperative autonomous players (robots). Many researchers have developed methods that do not require human designers to define specific behaviors of players for each problem. The work reported in this chapter focuses on the techniques of evolutionary computation, which has been regarded as one of the most promising approaches to solving such complex problems. However, in using evolutionary computation for generating players performing tasks cooperatively, one faces fundamental and difficult decisions, including the one regarding the so-called credit assignment problem (Haynes et al., 1995). For example, if we can only evaluate the global performance of each team, how do we divide up the team's performance among the participating players? We believe that there are some correlations among design decisions, and therefore a comprehensive evaluation of them is essential, although several researchers have proposed evolutionary methods for evolving teams performing specific tasks.

This chapter is organized as follows. In Section 2, we list three fundamental decisions and possible options in each decision in designing a method for evolving a cooperative team. We find that there are 18 typical combinations available. Then, in Section 3, we describe the ultimately simplified soccer game played on a one-dimensional field as a testbed for comparative evaluation of these 18 candidate methods. Section 4 reports on the results of the comparative evaluation of these methods, and Section 5 summarizes the work.

2. Methods for Evolving a Team

In general, three fundamental decisions are necessary when one designs an evolutionary computation method for generating players performing tasks cooperatively, and there may be several combinations of the options in these decisions.

The first decision is: How many evolving populations are there? The answer is derived by considering whether or not the population structure depends on the number of teams in the game, or the number of player roles in the game (Fig. 1). Suppose that the game is played by 2 teams each consisting of 3 players. We can assume an evolutionary computation with 2 populations corresponding 2 teams, with 3 populations corresponding 3 players, or with 6 populations corresponding to 2 teams of 3 players. So, the typical options for the number of

the populations are 1, R , T and $T \cdot R$ (T : number of teams in the game, R : number of the player roles in the team).

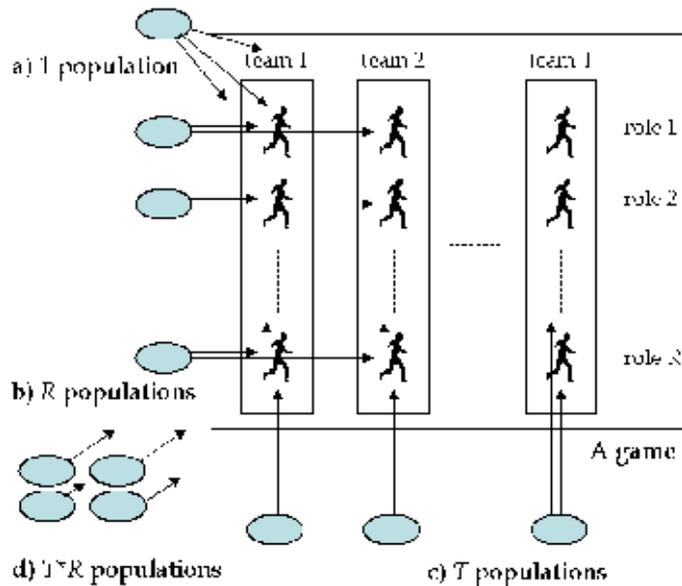


Figure 1. The four options for the population structure. a) The population represents all player roles in all teams. b) Each population represents one player role in all teams. c) Each population represents all player roles in each team. d) Each population represents one player role in each team

The second decision is: What does each individual (genome) represent? Typical options are a player and a team. In the case where each genome represents a player, there can be two further options: all players in the team share one genome ("homogeneous players") or all players are represented by different genomes ("heterogeneous players"). In the case where each genome represents a team, there can be two further options: whether or not the roles of the players represented in each genome are fixed. In the case where the roles of the player are fixed, for example, if a part of a genome represents a defender in the game, this part always represents a defender.

The third decision is: How is the fitness function evaluated? One option is that fitness is evaluated for a team as a whole. In this case, if each genome represents a player, each player in a team is supposed to have the same fitness. The other option is that the fitness is evaluated for each player directly or indirectly. Direct evaluation of players in a cooperative team is sometimes a very difficult task, as in general altruistic behavior is important or essential in the establishment and maintenance of cooperation in population. Some methods for indirect evaluation have been proposed (Miconi, 2001). We adopt a method as this option in which the fitness of a player is defined as the decrease in the fitness of the team when the player is replaced by a predefined "primitive player" who has a minimum set of behavior rules.

Therefore, there could be 18 available combinations for evolving players performing tasks cooperatively, as shown in Table 1.

Population structure			Each genome represents		Unit of fitness evaluation is		Code name	
depends on		Number of populations						
T?	R?							
No	No	1	a player	heterogeneous players	a player	by direct evaluation	1-PHe-PD	
						by indirect evaluation	1-PHe-PI	
				a team (same fitness in a team)		1-PHe-T		
				homogeneous players	a team (same fitness in a team)	1-PHo-T		
		a team	fixed player-roles	a team	1-TFi-T			
			unfixed player-roles	a team	1-TUn-T			
	Yes	R	R	a player	heterogeneous players	a player	by direct evaluation	R-PHe-PD
							by indirect evaluation	R-PHe-PI
					a team (same fitness in a team)		R-PHe-T	
Yes	No	T	a player	heterogeneous players	a player	by direct evaluation	T-PHe-PD	
						by indirect evaluation	T-PHe-PI	
				a team (same fitness in a team)		T-PHe-T		
				homogeneous players	a team (same fitness in a team)	T-PHo-T		
		a team	fixed player-roles	a team	T-TFi-T			
			unfixed player-roles	a team	T-TUn-T			
	Yes	T*R	T*R	a player	heterogeneous players	a player	by direct evaluation	TR-PHe-PD
							by indirect evaluation	TR-PHe-PI
					a team (same fitness in a team)		TR-PHe-T	

Table 1. Classification of the methods for evolving a team (T , number of teams in a game; R , number of player roles in a team)

Many researchers treated this issue, although most of them focused on one or two methods of 18 combinations. Some significant studies are classified into one of these combinations as follows: 1-PHo-T is the simplest method, in which there is one population and all players in a team share one genome. Quinn et al. (Quinn et al., 2002) adopted this method, and successfully evolved robots that work as a team, adopting and maintaining distinct but interdependent roles, based on their relative positions in order to achieve a formation movement task, although they were homogenous. Miconi (Miconi, 2001) adopted 1-PHe-PI, in which the fitness of each individual was determined as the decrease in fitness when that individual was not present in the team in the context of on-line evolution. Luke (Luke, 1998) evolved teams of soccer players through an adapted version of genetic programming: homogenous teams (1-PHo-T) and heterogeneous teams (1-TFi-T). Potter and De Jong (Potter & De Jong, 1994) proposed cooperative evolutionary algorithms, which can be classified into R-PHe-T, and tested them in the domain of function optimization, in which each population contained values for one parameter, and the fitness of a parameter was obtained by combining the current best parameters of the remaining populations. Our previous study (Asai & Arita, 2003) compared [1-PHe-PD], [1-PHe-T], [1-TFi-T], [R-PHe-PD] and [R-PHe-T] using a multirobot model in which not only control of behaviors, but also morphology (including selection and the arrangement of sensors/motors), evolved via ontogenesis.

3. Ultimately Simplified Soccer Game

The ultimately simplified soccer game is defined as a testbed for comparative evaluation of these 18 candidate methods. It is a 2 vs 2 player game played on a one-dimensional cellular field ($T=R=2$ in Fig. 1), as shown in Fig. 2 (field: 1-20). The players are homogeneous except in their starting positions (left team: player 1 (field 8), player 2 (field 5); Right team: player 1 (field 13), player 2 (field 16)), and each player makes a run, dribbles the ball, makes a shot at goal, or passes the ball to the other player of their own team. One of the actions is decided on based on the relative locations of all players and the ball (72 patterns). Action is taken in turn between 2 teams. Each step in the game is composed of 4 actions by all players.

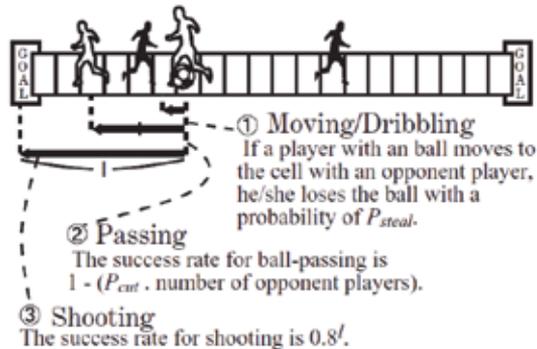


Figure 2. The ultimately simplified soccer game

Multiple players can not be in one cell. The ball is always in a cell where a player resides. A moving action of a player with the ball means dribbling. Players move to either of the neighboring cells, but when a player moves to a cell with another player, the neighboring player is skipped over (a player cannot skip more than one other player). In this case, if the players are in opposite teams and one of them has the ball, the ball moves to the other player with a set probability (P_{steal}). If there is an opponent player between the passer and the receiver, the ball-passing becomes a failure with a set probability (P_{cut}), and in this case the ball moves to the cell where the opponent player resides. The success rate for shooting is antiproportional to the length between the player's position and the goal irrespective of the presence of the opposing players. If a goal is scored, the game restarts with the initial player locations. If there is a failure, the game restarts after the ball moves to the opposite player nearer to the goal post.

We expect two types of altruistic behavior which could lead to the emergence of cooperation in the game. One is passing the ball to the other player in the same team instead of dribbling the ball or taking a shot at the goal. The other type is running in the direction away from the goal. The former type of altruistic behavior is analyzed in Section 4.3.

4. Evaluation

4.1 Expression of the Players

Each player selects next action deterministically based on the positional relationship of the players and the ball. In doing so, two opponent players are not distinguished. So to be precise, the genetic information of each player decides the next action of that player based on one of 48 patterns, where each pattern is associated with one of the four actions:

running/dribbling to the right; running/dribbling to the left; feeding (passing) the ball to the other player of their own team; taking a shot at goal. Therefore, each player is represented by 96 bits of genetic information.

4.2 Evaluation Setting

The evaluation is conducted in two steps: an evolution step and an evaluation step. In the evolution step, populations are evolved for 2000 generations using 18 methods independently. Each population has 40 individuals in all methods. A round-robin tournament of an ultimately-simplified game of 200 steps is held to evaluate the fitness in each generation.

The parameters P_{steal} and P_{cut} are set to 0.8 and 0.4, respectively in both steps. These parameters were determined based on preliminary experiments mainly using [T-Pho-T], [T-Tfi-T] and [TR-Phe-T]. The evolution of the players depended significantly on both parameters. In short, a large P_{steal} or a small P_{cut} evolved the passer-type players. In contrast, a small P_{steal} or a large P_{cut} evolved the dribbler-type players. We found that the above settings could generate many different kinds of players.

With the <team-evaluated> option, the fitness is calculated as the number of goals the team scored minus the number of goals the opponent team scored. With the <direct-player-evaluated> option, the fitness is calculated as the number of goals the player scored minus the opponent team's goals divided by 2. Then tournament selection (repeatedly selecting the individuals with a higher fitness as parents by comparing two randomly chosen individuals), crossover with a 60% probability and one-point mutation with a 3% probability are adopted as genetic operators. With the <indirect-player-evaluated> option, we use a primitive player designed a priori as follows. When a player keeps the ball, if they are behind the other team player they pass the ball to the other player, otherwise they shoot. When a player does not keep the ball, if they are behind the other team player, they move back, otherwise they move toward the goal. In the evaluation step, the best team is selected in each of the last 50 generations in the evolution step, and selected 50 times 18 selected teams conduct another round-robin tournament of 1000 step games.

4.3 Evaluation Results

Fig. 3 shows the winning ratio of the teams evolved by 18 methods, each of which is the average winning ratio of the best 10 teams from 50 teams in the all-play-all tournament described above. Table 2 (the left-hand column in the results) also shows these results. Each pair of bars in Fig. 3 shows the results of the strategies with same options in genome representation and fitness evaluation except for the population structure option (upper white bars, <1/R-populations> options; lower black bars, <T/T*R-populations> options).

It can be seen that the top three methods in this evaluation are <1-population, team-represented with fixed player-roles, team-evaluated>, <T*R-populations, heterogeneous-player-represented, team-evaluated>, and <1-population, homogeneous-player-represented, team-evaluated>. The winning ratios are 74.6%, 74.1% and 73.5%, respectively. An additional evaluation using a team consisting of two primitive players showed that its winning ratio was 16.0%. This ratio could be a measure for the performance of these methods.

Regarding the population structure, the $\langle 1/R\text{-populations} \rangle$ option performed better than the $\langle T/T^*R\text{-populations} \rangle$ option in general. This might be because of ill-balanced evolution, over-specialization, or circularity. The adoption of an asymmetric game as a testbed would make this tendency weaker. Regarding genome representation, the $\langle \text{homogeneous-player-represented} \rangle$ option performed well in general. Also, the $\langle \text{team-represented with fixed player-roles} \rangle$ option performed well, although the $\langle \text{team-represented with unfixed player-roles} \rangle$ option performed badly. Regarding fitness evaluation, the $\langle \text{team-evaluated} \rangle$ option performed well in general, as the fact that five of the top six methods adopt this option has shown. The performance of the $\langle \text{indirect-player-valuation} \rangle$ option depended largely on the other options.

We observed an interesting separation of roles between the two players in the teams with a high winning ratio. For example, in some teams the forward player tended to play near the goal and the backward player tended to move in order to intercept the ball, and in some teams both players seemed to use man-to-man defence.

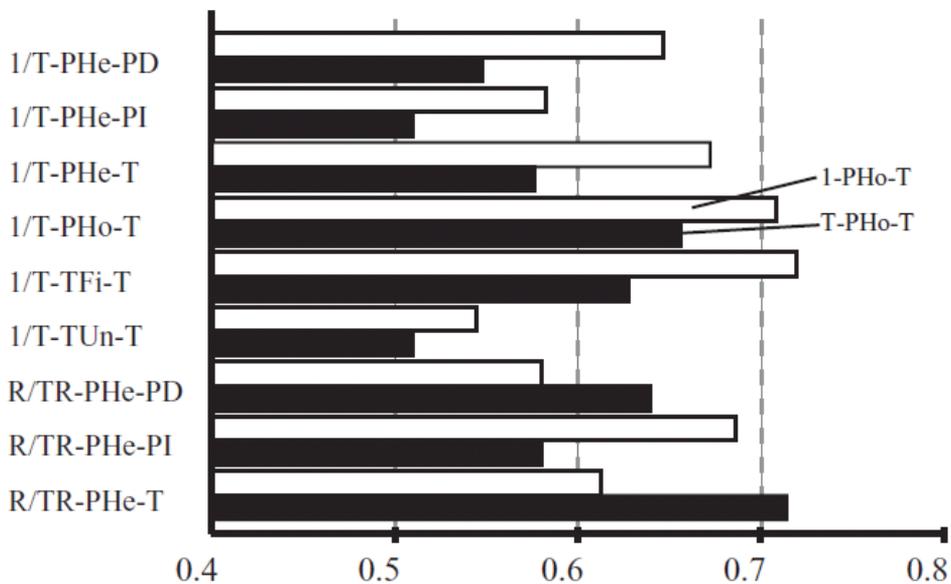


Figure 3. The average winning ratio of the best 10 teams evolved by each of 18 methods

Next we examined the relationship between altruistic behavior which could lead to cooperative behavior and the winning ratio. Here we focus on the following behavior pattern. A player with the ball passes to the other team player, who receives the ball without being intercepted and then successfully shoots a goal immediately or after dribbling. We termed this series of actions as an “assisted goal” (Fig. 4). Table 2 shows the assist ratio, which is the ratio of assisted goals to all goals, and the winning ratio of the teams evolved by 18 methods. We see from this table that good performing teams also have a tendency to have a high assist ratio. In contrast, it is not necessarily the case that teams with a high assist ratio have a tendency to have a high winning ratio. This means that assisting behaviour defined above is a necessary requirement for the teams to perform well.

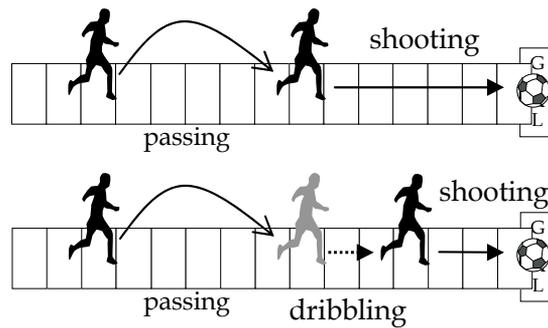


Figure 4. Two types of “assisted goals”

Code name	Results			
	Winning ratio	Rank	Assist ratio	Rank
1-PHe-PD	0.673	7	0.146	15
1-PHe-PI	0.609	11	0.289	10
1-PHe-T	0.699	5	0.310	9
1-PHo-T	0.735	3	0.390	5
1-TFi-T	0.746	1	0.342	7
1-TUn-T	0.571	16	0.336	8
R-PHe-PD	0.607	12	0.109	16
R-PHe-PI	0.713	4	0.503	1
R-PHe-T	0.639	10	0.402	3
T-PHe-PD	0.574	15	0.080	17
T-PHe-PI	0.536	17	0.391	4
T-PHe-T	0.603	14	0.242	12
T-PHo-T	0.683	6	0.226	13
T-TFi-T	0.654	9	0.260	11
T-TUn-T	0.536	18	0.214	14
TR-LPHe-LD	0.666	8	0.077	18
TR-PHe-PI	0.607	13	0.388	6
TR-PHe-T	0.741	2	0.116	2

Table 2. Average winning ratio and assist ratio

It is a remarkable fact that the <indirect-player-evaluated> option made the assist ratio higher. For this option, we adopted a method in which the fitness of a player is the decrease in the fitness of team when the player is replaced by a primitive player. This method should generate a strong interaction between two players because it tends to result in large decrease when the player is replaced. Therefore, the teams generated by the indirect evaluation method have a higher assist ratio despite having a relatively low winning ratio.

5. Conclusion

This chapter has focused on the methods for evolving a cooperative team by conducting a comparative evaluation of 18 methods. We have found that some methods performed well, while there are complex correlations among design decisions. Also, further analysis has

shown that cooperative behavior can be evolved, and can be a necessary requirement for the teams to perform well even in such a simple game. These results could provide insights into the evolutionary design of multi-robot systems working in cooperative tasks.

Evolutionary computation mimics the biological evolution of living organisms. We believe that the credit assignment problem, which is the focus of this chapter, could be solved more efficiently by developing the biological knowledge on the mechanism of the evolution of altruistic behavior, specifically on multi-level selection, a modern version of group selection (Ichinose & Arita, 2008).

6. References

- Asai Y. & Arita T. (2003). Coevolution of Morphology and Behavior of Robots in a Multi-agent Environment, *Proceedings of the SICE 30th Intelligent System Symposium*, pp. 61-66, Tokyo, March 2003, The Society of Instrument and Control Engineers, Tokyo.
- Haynes, T. ; Sen, S ; Schoenefeld, D. & Wainwright, R. (1995). Evolving a Team, *Working Notes for the AAAI Symposium on Genetic Programming*, pp. 23-30, Cambridge, MA November 1995, AAAI Press, Menlo Park, CA
- Ichinose G. Arita, T. (2008). The Role of Migration and Founder Effect for the Evolution of Cooperation in a Multilevel Selection Context, *Ecological Modelling*, Vol. 210, Issue 3, January 2008, pp. 221-230, ISSN 0304-3800
- Luke, S. (1998). Genetic Programming Produced Competitive Soccer Softbot Teams for RoboCup97, *Proceedings of the Third Annual Genetic Programming Conference (GP)*, pp. 204-222, ISBN 1-55860-548-7, Madison, July 1998, Morgan Kaufmann Publishers, San Francisco
- Miconi, T. (2001). A Collective Genetic Algorithm, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 876-883, ISBN 1558607749, San Francisco, CA, July 2001, Morgan Kaufmann Publishers, San Francisco
- Potter, M. A. & De Jong, K. A. (1994). Cooperative Coevolutionary Approach to Function Optimization', *Proceedings of the Third Parallel Problem Solving from Nature (PPSN)*, pp. 249-257, Jerusalem, October 1994, Springer-Verlag, New. York
- Quinn, M.; Smith, L.; Mayley G. & Husbands, P. (2002). Evolving Formation Movement for a Homogeneous Multi-Robot System: Teamwork and Role Allocation with Real Robots, *Cognitive Science Research Papers (CSRP)*, 515, School of Cognitive and Computing Sciences, University of Sussex, Brighton

An Adaptive Penalty Method for Genetic Algorithms in Constrained Optimization Problems

Helio J. C. Barbosa and Afonso C. C. Lemonge
Laboratório Nacional de Computação Científica, LNCC/MCT
Universidade Federal de Juiz de Fora, UFJF
Brazil

1. Introduction

Inspired by the Darwinian principles of evolution, genetic algorithms (GAs) (Holland, 1975, Goldberg, 1989) are widely used in optimization. GAs encode all the variables corresponding to a candidate solution (the phenotype) in a data structure (the genotype) and maintain a population of genotypes which is evolved mimicking Nature's evolutionary process. Individuals are selected for reproduction in a way that better performing solutions have a higher probability of being selected. The genetic material contained in the chromosome of such "parent" individuals is then recombined and mutated, by means of crossover and mutation operators, giving rise to offspring which will form a new generation of individuals. The process is repeated for a given number of generations or until some stopping criteria are met. GAs are search algorithms which can be directly applied to unconstrained optimization problems, UOP(f, S), where one seeks for an element x belonging to the search space S , which minimizes (or maximizes) the real-valued objective function f . In this case, the GA usually employs a fitness function closely related to f . However, practical optimization problems often involve several types of constraints and one is led to a constrained optimization problem, COP(f, S, b), where an $x \in S$ that minimizes (or maximizes) f in S is sought with the additional requirement that the boolean function $b(x)$ (which includes all constraints of the problem) is evaluated as true. Elements in S satisfying all constraints are called feasible.

The straightforward application of GAs to COPs is not possible due to the additional requirement that a set of constraints must be satisfied. Difficulties may arise as the objective function may be undefined for some (or all) infeasible elements, and an informative measure of the degree of infeasibility of a given candidate solution may not be easily defined. Finally, for some real-world problems, the check for feasibility can be more expensive than the computation of the objective function itself.

As a result, several techniques have been proposed and compared in the literature in order to enable a GA to tackle COPs. Those techniques can be classified either as *direct* (feasible or

interior), when only feasible elements in S are considered, or as *indirect* (exterior), when both feasible and infeasible elements are used during the search process.

Direct techniques include: a) the design of special *closed* genetic operators, b) the use of special decoders, c) repair techniques, and d) "death penalty".

In special situations, closed genetic operators (in the sense that when applied to feasible parents they produce feasible offspring) can be designed if enough domain knowledge is available (Shoenauer & Michalewicz, 1996). Special decoders (Koziel & Michalewicz, 1999) - that always generate feasible individuals from any given genotype have been devised, but no applications considering non-linear implicit constraints have been published.

Repair methods (Liepins & Potter, 1996, Orvosh & Davis, 1994) use domain knowledge in order to move an infeasible offspring into the feasible set. However, there are situations when it is very expensive, or even impossible, to construct such a repair operator, drastically reducing the range of applicability of repair methods. The design of efficient repair methods constitutes a formidable challenge, specially when non-linear implicit constraints are present.

Discarding any infeasible element generated during the search process ("death penalty") is common practice in non-population optimization methods. Although problem independent, no consideration is made for the potential information content of any infeasible individual.

Summarizing, direct techniques are problem dependent (with the exception of the "death penalty") and actually of extremely reduced practical applicability.

Indirect techniques include: a) the use of Lagrange multipliers (Adeli & Cheng, 1994), which may also lead to the introduction of a population of multipliers and to the use of the concept of coevolution (Barbosa, 1999), b) the use of fitness as well as constraint violation values in a multi-objective optimization setting (Surry & Radcliffe, 1997), c) the use of special selection techniques (Runarsson & Yao, 2000), and d) "lethalization": any infeasible offspring is just assigned a given, very low, fitness value (Kampen et al., 1996).

For other methods proposed in the evolutionary computation literature see (Shoenauer & Michalewicz, 1996, Michalewicz & Shoenauer, 1996, Hinterding & Michalewicz, 1998, Koziel & Michalewicz, 1998, Kim & Myung, 1997), references therein, and the repository <http://www.cs.cinvestav.mx/constraint/>, maintained by C.A.C. Coello.

Methods to tackle COPs which require the knowledge of constraints in explicit form have thus limited practical applicability. This fact, together with simplicity of implementation are perhaps the main reasons why penalty techniques, in spite of their shortcomings, are the most popular ones.

Penalty techniques, originating in the mathematical programming community, range from simple schemes (like "lethalization") to penalty schemes involving from one to several parameters. Those parameters can remain constant (the most common case) or be dynamically varied along the evolutionary process according to an exogenous schedule or an adaptive procedure. Penalty methods, although quite general, require considerable domain knowledge and experimentation in each particular application in order to be effective.

In previous work (Barbosa & Lemonge, 2002, Lemonge & Barbosa, 2004), the authors developed a general penalty method for GAs which (i) handles inequality as well as equality constraints, (ii) does not require the knowledge of the explicit form of the constraints as a function of the decision/design variables, (iii) is free of parameters to be set by the user, (iv) can be easily implemented within an existing GA code and (v) is robust.

In this adaptive penalty method (APM), in contrast with approaches where a single penalty parameter is used, an adaptive scheme automatically sizes the penalty parameter corresponding to *each* constraint along the evolutionary process.

In the next section the penalty method and some of its traditional implementations within genetic algorithms are presented. In Section 3 the APM is revisited and some variants are proposed, Section 4 presents numerical experiments with several test-problems from the literature and the paper closes with some conclusions.

2. Penalty Methods

A standard constrained optimization problem in R^n can be written as the minimization of a given objective function $f(x)$, where $x \in R^n$ is the vector of design/decision variables, subject to inequality constraints $g_p(x) \geq 0$, $p = 1, 2, \dots, \bar{p}$ as well as equality constraints $h_q(x) = 0$, $q = 1, 2, \dots, \bar{q}$. Additionally, the variables may be subject to bounds $x_i^L \leq x_i \leq x_i^U$, but this type of constraint is trivially enforced in a GA.

Penalty techniques can be classified as *multiplicative* or *additive*. In the multiplicative case, a positive penalty factor $p(v(x), T)$ is introduced in order to amplify (in a minimization problem) the value of the fitness function of an infeasible individual. One would have $p(v(x), T) = 1$ for a feasible candidate solution x and $p(v(x), T) > 1$ otherwise. Also, $p(v(x), T)$ increases with the "temperature" T and with constraint violation. An initial value for the temperature is required as well as the definition of a schedule whereby T grows with the generation number. This type of penalty has received much less attention in the evolutionary computation community than the additive type. In the additive case, a penalty functional is added to the objective function in order to define the fitness value of an infeasible element. They can be further divided into: (a) *interior* techniques, when a barrier functional $B(x)$ -which grows rapidly as x approaches the boundary of the feasible domain- is added to the objective function

$$F_k(x) = f(x) + \frac{1}{k} B(x)$$

and (b) *exterior* techniques, where a penalty functional is introduced

$$F_k(x) = f(x) + kP(x) \quad (1)$$

such that $P(x) = 0$, if x is feasible, and $P(x) > 0$ otherwise (for minimization problems). In both cases, as $k \rightarrow \infty$, the sequence of minimizers of the UOP (F_k, S) converges to the solution of the COP (f, S, b) .

At this point, it is useful to define the amount of violation of the j -th constraint by the candidate solution $x \in R^n$ as

$$v_j(x) = \begin{cases} |h_j(x)|, & \text{for an equality constraint,} \\ \max\{0, -g_j(x)\} & \text{otherwise} \end{cases}$$

It is also common to design penalty functions that grow with the vector of violations $v(x) \in R^m$ where $m = \bar{p} + \bar{q}$ is the number of constraints to be penalized. The most popular penalty function is given by

$$P(x) = k \sum_{j=1}^m (v_j(x))^\beta \quad (2)$$

where k is the penalty parameter and $\beta = 2$. Although it is easy to obtain the unconstrained problem, the definition of the penalty parameter k is usually a time-consuming problem-dependent trial-and-error process.

Le Riche et al. (1995) present a GA based on two-level of penalties, where two fixed penalty parameters k_1 and k_2 are used independently in two different populations. The idea is to create two sets of candidate solutions where one of them is evaluated with the parameter k_1 and the other with the parameter k_2 . With $k_1 \neq k_2$, there are two different levels of penalization and there is a higher chance of maintaining feasible as well as infeasible individuals in the population and to get offspring near the boundary between the feasible and infeasible regions. The strategy can be summarized as: (i) create $2 \times pop$ individuals randomly (pop is the population size); (ii) evaluate each individual considering each penalty parameter and create two ranks; (iii) combine the two ranks in a single one with size pop ; (iv) apply genetic operators; (v) evaluate new offspring ($2 \times pop$), using both penalty parameters and repeat the process.

Homaifar et al. (1994) proposed a penalty strategy with multiple coefficients for different levels of violation of each constraint. The fitness function is written as

$$F(x) = f(x) + \sum_{j=1}^m k_{ij} (v_j(x))^2$$

where i denotes one of the l levels of violation defined for the j -th constraint. This is an attractive strategy because, at least in principle, it allows for a good control of the penalization process. The weakness of this method is the large number, $m(2l+1)$, of parameters that must be set by the user for each problem.

Joines & Houck (1994) proposed that the penalty parameters should vary dynamically along the search according to an exogenous schedule. The fitness function $F(x)$ was written as in (1) and (2) with the penalty parameter, given by $k = (C \times t)^\alpha$, increasing with the generation number t . They used both $\beta = 1, 2$ and suggested the values $C = 0.5$ and $\alpha = 2$.

Powell & Skolnick (1994), proposed a method of superiority of feasible points where each candidate solution is evaluated by the following expression:

$$F(x) = f(x) + r \sum_{j=1}^m y_j(x) + \theta(t, x)$$

where r is a constant. The main assumption is that any feasible solution is better than any infeasible solution. This assumption is enforced by a convenient definition of the function $\theta(t, x)$. This scheme can be modified by the introduction of the tournament selection proposed in (Deb, 2000), coupled with the fitness function:

$$F(x) = \begin{cases} f(x), & \text{if } x \text{ is feasible,} \\ f_{max} + \sum_{j=1}^m y_j(x), & \text{otherwise} \end{cases}$$

where f_{max} is the function value of the worst feasible solution. As observed in the expression above, if a solution is infeasible the value of its objective function is not considered in the computation of the fitness function, instead, f_{max} is used. However, Deb's constraint handling scheme (Deb, 2000) -which also uses niching and a controlled mutation- works very well for his real-coded GA, but not so well for his binary-coded GA. Among the many suggestions in the literature (Powell & Skolnick, 1993, Michalewicz & Shoenauer, 1996, Deb, 2000) some of them -more closely related to the work presented here- will be briefly discussed in the following.

2.1 Adaptive penalties

A procedure where the penalty parameters change according to information gathered during the evolution process was proposed by Bean & Hadj-Alouane (Bean & Alouane, 1992). The fitness function is again given by (1) and (2) but with the penalty parameter $k = \lambda(t)$ adapted at each generation by the following rules:

$$\lambda(t+1) = \begin{cases} (1\beta_1)\lambda(t), & \text{if } b^i \in F \text{ for all } t-g+1 \leq i \leq t \\ \beta_2\lambda(t), & \text{if } b^i \notin F \text{ for all } t-g+1 \leq i \leq t \\ \lambda(t) & \text{otherwise} \end{cases}$$

where b^i is the best element at generation i , F is the feasible region, $\beta_1 \neq \beta_2$ and $\beta_1, \beta_2 > 1$. In this method, the penalty parameter of the next generation $\lambda(t+1)$ decreases when all best elements in the last g generations are feasible, increases if all best elements are infeasible, and otherwise remains without change. The fitness function is written as:

$$F(x) = f(x) + \lambda(t) \left[\sum_{i=1}^n g_i^2(x) + \sum_{j=1}^p |h_j(x)| \right]$$

Schoenauer & Xanthakis (1993), presented a strategy that handles constrained problems in stages: (i) initially, a randomly generated population is evolved considering only the first constraint until a certain percentage of the population is feasible with respect to that constraint; (ii) the final population of the first stage of the process is used in order to optimize with respect to the second constraint. During this stage, the elements that had violated the previous constraint are removed from the population, (iii) the process is repeated until all the constraints are processed. This strategy becomes less attractive as the number of constraints grows and is potentially dependent on the order in which the constraints are processed.

The method proposed by Coit et al. (1996), uses the fitness function:

$$F(x) = f(x) + (F_{feas}(t) - F_{all}(t)) \sum_{j=1}^m (v_j(x)/v_j(t))^\alpha$$

where $F_{all}(t)$ corresponds to the best solution (without penalty), until the generation t , F_{feas} corresponds to the best feasible solution, and α is a constant. The fitness function is also written as:

$$F(x) = f(x) + (F_{feas}(t) - F_{all}(t)) \sum_{j=1}^m (v_j(x)/NFT(t))^\alpha$$

where NFT (Near Feasibility Threshold) defines the threshold distance, set by the user, between the feasible and infeasible regions of the search space.

A variation of the method proposed by Coit et al. (1996) is presented in (Gen & Gheng, 1996a), where the fitness function is defined as:

$$F(x) = f(x) \left[1 - \frac{1}{n} \sum_{i=1}^n \left(\frac{\Delta b_i(x)}{b_i} \right) \right]$$

and $\Delta b_i(x) = \max[0, g_i(x) - b_i]$ refers to the constraint violation. Based upon the same idea, Gen and Cheng (1996b), proposed an adaptive penalty scheme where the fitness function is written as:

$$F(x) = f(x) \left[1 - \frac{1}{n} \sum_{i=1}^n \left(\frac{\Delta b_i(x)}{\Delta b_i^{max}} \right) \right]$$

where $\Delta b_i^{max} = \max[\varepsilon, \Delta b_i(x)]$ is the maximum violation of the constraint i and ε is a small positive value to avoid dividing by zero.

Hamida & Schoenauer (2000), proposed an adaptive scheme using: (i) a function of the proportion of feasible individuals in the population, (ii) a seduction/selection strategy to mate feasible and infeasible individuals, and (iii) a selection scheme to give advantage for a given number of feasible individuals.

3. The Adaptive Penalty Method

The adaptive penalty method (APM) was originally introduced in (Barbosa & Lemonge, 2002). In (Barbosa & Lemonge, 2003), the behavior of the penalty parameters was studied and further tests were performed. Finally, in (Lemonge & Barbosa, 2004.), a slight but important modification was introduced leading to its present form. The method does not require any type of user defined penalty parameter and uses information from the population, such as the average of the objective function and the level of violation of each constraint during the evolution.

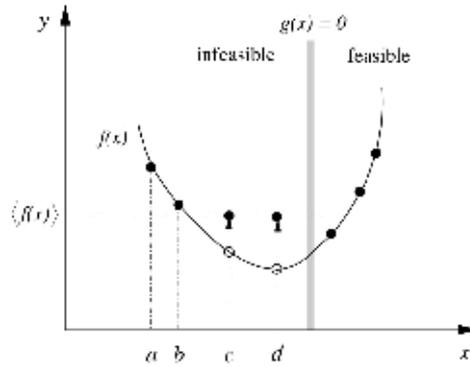


Figure 1. A pictorial description of the function \bar{f}

The fitness function proposed in APM is written as:

$$F(x) = \begin{cases} f(x), & \text{if } x \text{ is feasible} \\ \bar{f}(x) + \sum_{j=1}^m k_j v_j(x) & \text{otherwise} \end{cases}$$

where

$$\bar{f}(x) = \begin{cases} f(x), & \text{if } f(x) > \langle f(x) \rangle, \\ \langle f(x) \rangle & \text{otherwise} \end{cases} \tag{3}$$

and $\langle f(x) \rangle$ is the average of the objective function values in the current population. In the Figure 1 feasible as well as infeasible solutions are shown. Among the 4 infeasible solutions, the individuals c and d have their objective function values (represented by open circles) less than the average objective function and, according to the proposed method, have \bar{f} given by $\langle f(x) \rangle$. The solutions a and b have objective function values which are worse than the population average and thus have $\bar{f}(x) = f(x)$.

The penalty parameter is defined at each generation by:

$$k_j = |\langle f(x) \rangle| \frac{\langle v_j(x) \rangle}{\sum_{l=1}^m [\langle v_l(x) \rangle]^2} \quad (4)$$

and $\langle v_l(x) \rangle$ is the violation of the l -th constraint averaged over the current population. Denoting by pop the population size, one could also write

$$k_j = \frac{|\sum_{i=1}^{pop} f(x^i)|}{\sum_{l=1}^m \left[\sum_{i=1}^{pop} v_l(x^i) \right]^2} \sum_{i=1}^{pop} v_j(x^i) \quad (5)$$

The idea is that the values of the penalty coefficients should be distributed in a way that those constraints which are more difficult to be satisfied should have a relatively higher penalty coefficient.

Besides been tested by the authors, the APM has also been used by other researchers in different fields (Zavislak, 2004, Gallet, 2005, Obadage & Hampornchai, 2006).

4. Variants of the APM

The adaptive penalty method, as originally proposed, (i) computes the constraint violations v_j in the current population, and (ii) updates all penalty coefficients k_j at every generation. As a result, those coefficients undergo (potentially large) variations at every generation as shown in (Barbosa & Lemonge, 2003).

Despite its success, questions naturally arise concerning, for instance, the desirability of reducing the amplitude of those oscillations, and the possibility of reducing the computer time without compromising the quality of the results.

In the following, some possible variants of the APM will be considered.

4.1 The sporadic APM

One first variant of the APM corresponds to: (i) compute the constraint violations v_j in the current population, (ii) update the penalty coefficients k_j , but (iii) keep the penalty coefficients k_j fixed for f generations.

The gains in computer time are obvious: step (i) is performed only every f generations. Of course, one should investigate if convergence is negatively affected.

4.2 The sporadic APM with constraint violation accumulation

A second variant considered here corresponds to: (i) accumulate the constraint violations v_j for f generations, (ii) update the penalty coefficients k_j , and (iii) keep the penalty coefficients k_j fixed for f generations

The gains in computer time are very small, but the oscillations in the k_j values can be reduced.

4.3 The APM with monotonic penalty coefficients

As the penalty coefficients should (at least theoretically) grow indefinitely, another variant corresponds to the monotonic APM: no k_j is allowed to have its value reduced along the evolutionary process:

4.4 The APM with damping

Finally, one can think of reducing the oscillations in k_j by using a weighted average between the current value of k_j and the new value predicted by the method:

$$k_j^{(new)} = \theta k_j^{(new)} + (1 - \theta) k_j^{current}$$

where $\theta \in [0, 1]$.

4.5 A compact notation

A single notation can be introduced in order to describe the whole family of adaptive penalty techniques.

The numerical parameters involved are:

- The frequency f (measured in number of generations) of penalty parameter updating,
- The weight parameter $\theta \in [0, 1]$ used in the update formula for the penalty parameter.

Two additional features are: (i) constraint violations can alternatively be accumulated for all the f generations, or can be computed only every f generations, and (ii) the penalty parameter is free to vary along the process, or monotonicity will be enforced.

The compact notation $APM(f^{acc}, \theta_m)$ indicates a member of the family of methods where the penalty coefficients are updated every f generations, based on constraint violations which are accumulated (superscript acc) over those f generations, using the weight parameter θ , and enforcing monotonicity (subscript m). Thus, the original APM corresponds to $APM(1,1)$.

The five members of the APM family considered for testing here are:

Variant	Description
1	• The original APM: $APM(1,1)$
2	• The sporadic APM: $APM(f,1)$
3	• The sporadic APM with constraint violation accumulation: $APM(f^{acc},1)$
4	• The sporadic APM with constraint violation accumulation and smoothing: $APM(f^{acc}, \theta)$
5	• The sporadic APM with constraint violation accumulation, smoothing, and monotonicity enforcement: $APM(f^{acc}, \theta_m)$

In the experiments performed, the values $f = 100$ and $f = 500$ are used when 1000 and 5000 generations are employed, respectively. Also, $\theta = 0.5$ was adopted. Several examples from the literature are considered in the next section in order to test the performance of the variants considered with respect to the original APM.

5. Numerical experiments

Our baseline binary-coded generational GA uses a Gray code, rank-based selection, and elitism (the best element is always copied into the next generation along with 1 copy where one bit has been changed). A uniform crossover operator was applied with probability equal to 0.9 and a mutation operator was applied bit-wise to the offspring with rate $p_m = 0.004$.

5.1 Test 1 - The G-Suite

The first set of experiments uses the well known G1-G11 suite of test-problems. A complete description of each problem can be found, for example, in (Lemonge, & Barbosa, 2004). The G-Suite, repeatedly used as a test-bed in the evolutionary computation literature, is made up of different kinds of functions and involves constraints given by linear inequalities, nonlinear equalities and nonlinear inequalities. For the eleven problems, the population size is set to 100, 25 independent runs were performed, with the maximum number of generations equal to 1000 and 5000, and each variable was encoded with 25 bits. For the variant 5, the value of θ is set to 0.5. The Tables 1 and 2 present the values of the best, mean and the worst solutions found for each function with respect to the five variants of the APM. The values presented in **boldface** correspond to the better solution among the variants.

Test-problem	Best-known	Variant	Best	Mean	Worst
g01	-15	1	-14.9996145	-14.9845129	-14.8267331
		2	-14.9997983	-14.9988652	-14.9939536
		3	-14.9997216	-14.9876210	-14.8305302
		4	-14.9997582	-14.9986873	-14.9960541
		5	-14.9996321	-14.9979965	-14.9891883
g02	-0.8036191	1	-0.7789661	-0.6996669	-0.5728089
		2	-0.7873126	-0.7250099	-0.5998890
		3	-0.7925228	-0.7255508	-0.6244844
		4	-0.7846081	-0.7191555	-0.6307588
		5	-0.7838283	-0.7097919	-0.5855941
g03	-1.0005001	1	-0.9972458	-0.7779740	-0.5027824
		2	-0.9361403	-0.4968387	-0.0368749
		3	-0.9803577	-0.4463771	-0.1012515
		4	-0.9682280	-0.4168279	-0.0212638
		5	-0.7666858	-0.3919809	-0.1462019
g04	-30665.538671	1	-30665.3165485	-30578.5496875	-30387.6140423
		2	-30664.6156191	-30578.3703906	-30458.6292214
		3	-30663.9053552	-30577.5611718	-30432.1269937
		4	-30664.4464089	-30573.7182031	-30369.9518043
		5	-30662.5018601	-30591.6589843	-30358.6826419
g05	5126.4967140	1	5127.3606448	5343.2514134	5993.0123939
		2	5127.0852075	5341.0769391	5935.3076090
		3	5126.7785185	5323.8655104	5935.3019471
		4	5127.0852075	5290.0768500	5935.3076090
		5	5128.8444266	5378.8769735	6102.9022396
g06	-6961.8138755	1	-6957.5403309	-6913.0707583	-6868.6591021
		2	-6951.1888908	-6901.2575390	-6705.1840698
		3	-6958.1031284	-6879.5979882	-6296.5426371
		4	-6954.9453586	-6906.1789453	-6797.2374636
		5	-6961.4475438	-6805.2293359	-5237.1477535
g07	24.3062090	1	24.7768086	27.7708738	35.6097627
		2	24.8049743	28.0120724	28.0120724
		3	25.0308511	28.4575817	44.1698225
		4	24.5450354	27.8486413	36.8722275
		5	24.7255055	29.8150354	63.5219816
g08	-0.0958250	1	-0.0958250	-0.08768981	-0.0258067
		2	-0.0958250	-0.06476799	-0.0013001
		3	-0.0958250	-0.62599863	-0.0147534
		4	-0.0958250	-0.65938715	-0.0255390
		5	-0.0958250	-0.53489946	0.0123813

Test-problem	Best-known	Variant	Best	Mean	Worst
<i>g09</i>	680.6300573	1	680.7376315	682.0270142	690.4830475
		2	680.7722707	681.5344751	685.8184957
		3	680.7343755	681.4195117	683.1598659
		4	680.8243208	681.8727295	687.0344127
		5	680.6814564	681.4697436	683.2056847
<i>g10</i>	7049.2480205	1	7070.5636522	8063.2916015	8762.119390
		2	7237.0250892	10056.5608887	14688.684065
		3	7191.9045654	10165.4067383	14733.840629
		4	7217.4084043	9607.9555664	13715.794540
		5	7117.7173709	9322.0432129	12645.139817
<i>g11</i>	0.7499000	1	0.7523536	0.8585980	0.9932359
		2	0.7521745	0.8878919	0.9992905
		3	0.7521730	0.8879267	0.9992905
		4	0.7521745	0.8879346	0.9991157
		5	0.7501757	0.8822223	0.9954779

Table 1. Results of the G-Suite using 1000 generations

Test-problem	Best-known	Variant	Best	Mean	Worst
<i>g01</i>	-15	1	-14.9999800	-14.95263298	-14.6373581
		2	-14.9999914	-14.99962558	-14.9969186
		3	-14.9999826	-14.99953311	-14.9920359
		4	-14.9999910	-14.99998634	-14.9999264
		5	-14.9999915	-14.9999189	-14.9996256
<i>g02</i>	-0.8036191	1	-0.8015561	-0.77248005	-0.7261164
		2	-0.8025720	-0.76760471	-0.7328427
		3	-0.8023503	-0.77583072	-0.7401626
		4	-0.8030480	-0.77158681	-0.7752280
		5	-0.8025918	-0.77851344	-0.7306644
<i>g03</i>	-1.0005001	1	-1.0004896	-1.00040364	-0.9999571
		2	-1.0004634	-0.98604559	-0.6746467
		3	-1.0004630	-0.97507574	-0.3823454
		4	-1.0004602	-0.97900237	-0.5014565
		5	-1.0004553	-0.99997359	-0.9962939
<i>g04</i>	-30665.538671	1	-30665.523774	-30665.094688	-30661.761001
		2	-30665.536925	-30664.093203	-30643.260211
		3	-30665.537505	-30665.365000	-30664.505410
		4	-30665.533257	-30664.861094	-30660.745418
		5	-30665.520939	-30664.819062	-30660.360061

Test-problem	Best-known	Variant	Best	Mean	Worst
<i>g05</i>	5126.4967140	1	5127.3606448	5321.5713820	5993.0113312
		2	5126.7738829	5323.8496730	5935.3030129
		3	5126.7738829	5323.8495456	5935.3030129
		4	5126.7738829	5323.8496943	5935.3030129
		5	5128.8443865	5385.9205375	6102.8994189
<i>g06</i>	-6961.8138755	1	-6961.7960836	-6742.3431201	-1476.0027566
		2	-6961.7960836	-6961.7763086	-6961.7592083
		3	-6961.7960836	-6961.7720312	-6961.7592083
		4	-6961.7960836	-6961.7763086	-6961.7592083
		5	-6961.7960837	-6961.7752278	-6961.7592083
<i>g07</i>	24.3062090	1	24.4162533	26.4573583	30.7900065
		2	24.6482626	27.2199523	39.8419615
		3	24.5543275	26.9851276	38.2220049
		4	24.7315254	27.8377833	51.1675542
		5	24.4148575	26.3590182	29.5413519
<i>g08</i>	-0.0958250	1	-0.0958250	-0.0876899	-0.0258094
		2	-0.0958250	-0.0565442	-0.0010461
		3	-0.0958250	-0.0649342	-0.0010460
		4	-0.0958250	-0.0660025	-0.0204120
		5	-0.0958250	-0.0632326	-0.0197011
<i>g09</i>	680.6300573	1	680.6334486	680.7850952	681.1733250
		2	680.6673020	680.8470361	681.7406035
		3	680.7122589	680.8832129	681.3188377
		4	680.6505355	680.8678125	681.3625388
		5	680.6527662	680.8803711	681.6149132
<i>g10</i>	7049.2480205	1	7205.1435740	8392.399951	10349.909364
		2	7064.1563375	10108.975371	15407.413766
		3	7062.6060743	9953.722324	14200.173044
		4	7064.5428438	8554.065429	13156.5681862
		5	7064.3344362	10079.815684	14387.7769407
<i>g11</i>	0.7499000	1	0.7523536	0.8556172	0.9468393
		2	0.7521745	0.8834432	0.9903447
		3	0.7521745	0.8834432	0.9903447
		4	0.7521745	0.8845256	0.9937399
		5	0.7501757	0.8761453	0.9944963

Table 2. Results of the G-Suite using 5000 generations

Observing the results presented in Tables 1 and 2 one can see that the original APM found more often the best solutions. Among the new variants, the variants 3 and 5 were the most successful. As the performance of variant 4 was not as good as that of variants 3 and 5, one may suspect that, at least for this set of problems, the inclusion of the damping procedure was detrimental.

The graphics from the figures 2 to 25 show the values of the coefficients k_j along the 1000 generations performed for the functions *g06*, *g09*, *g04* and *g01*. For the Variant 1 are drawn

two graphics where the second one corresponds to a logarithmic scale for the coefficient k_j in the y -axis.

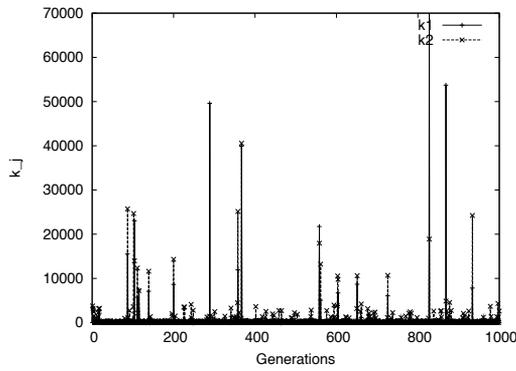


Figure 2. Variant 1 - Test-problem $g06$

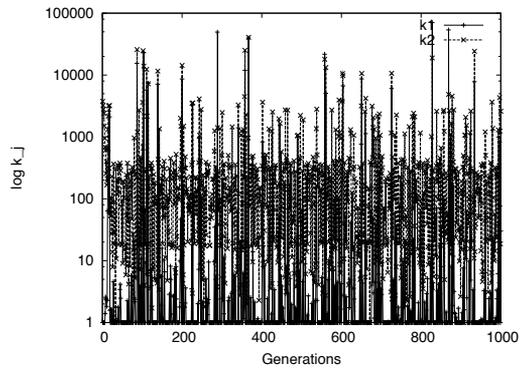


Figure 3. Variant 1 - Test-problem $g06$ (log scale for y -axis)

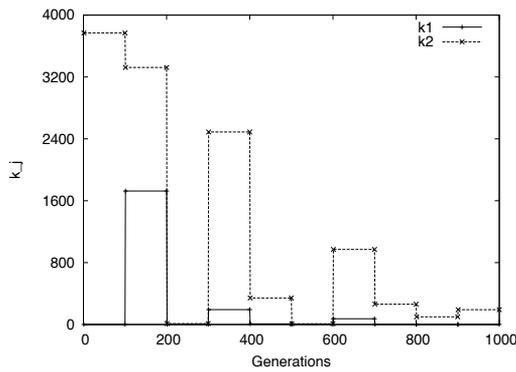


Figure 4. Variant 2 - Test-problem $g06$

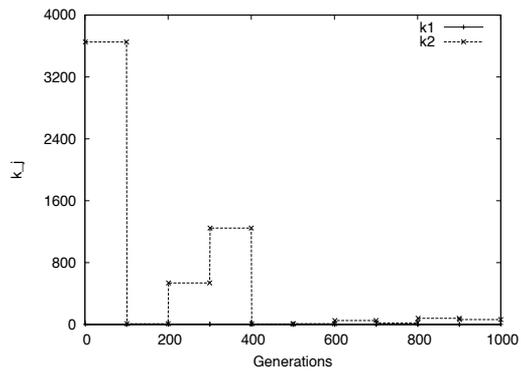


Figure 5. Variant 3 - Test-problem $g06$

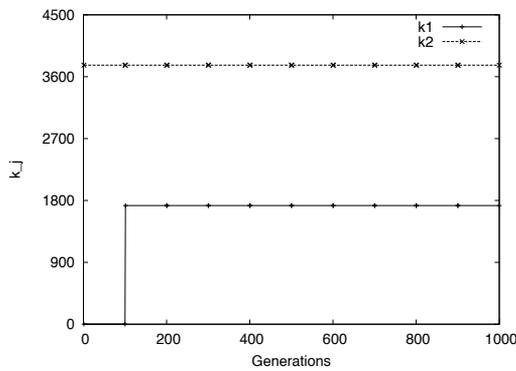


Figure 6. Variant 4 - Test-problem $g06$

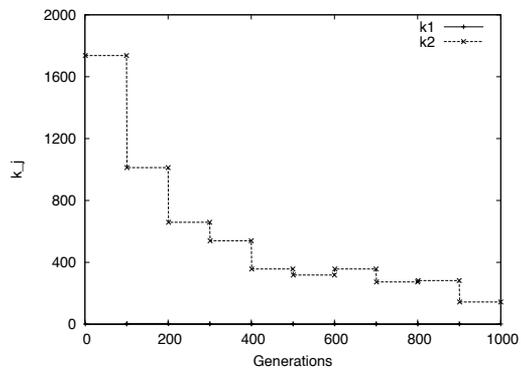


Figure 7. Variant 5 - Test-problem $g06$

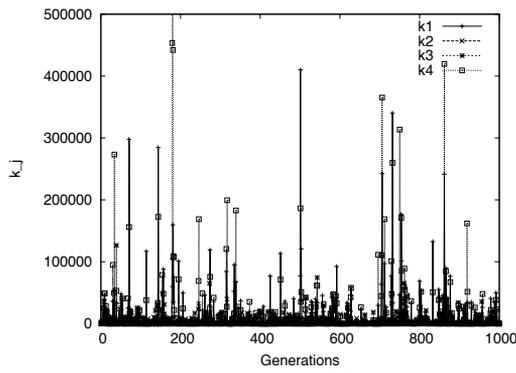


Figure 8. Variant 1 - Test-problem g_{09}

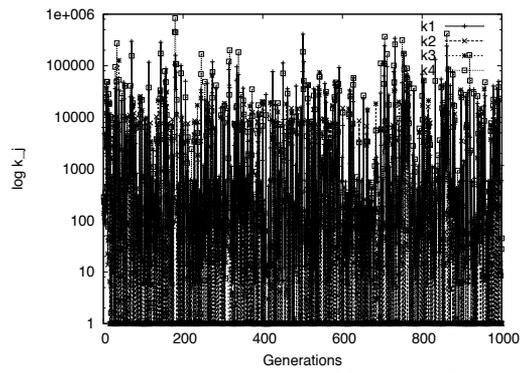


Figure 9. Variant 1 - Test-problem g_{09} (log scale for y -axis)

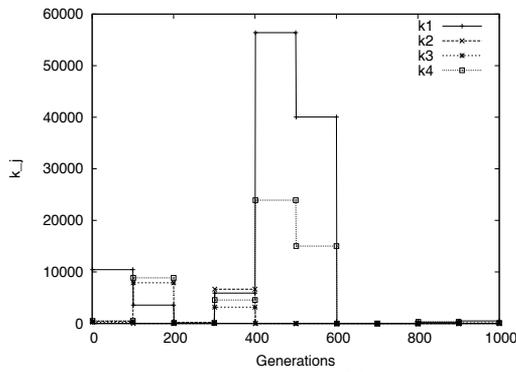


Figure 10. Variant 2 - Test-problem g_{09}

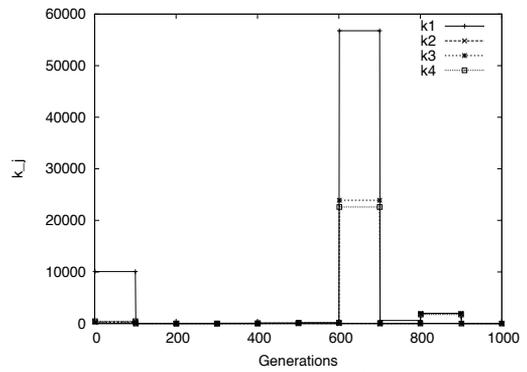


Figure 11. Variant 3 - Test-problem g_{09}

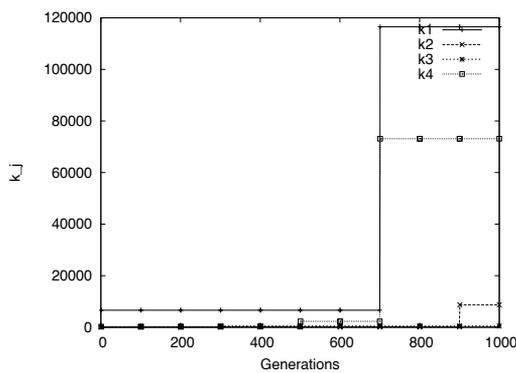


Figure 12. Variant 4 - Test-problem g_{09}

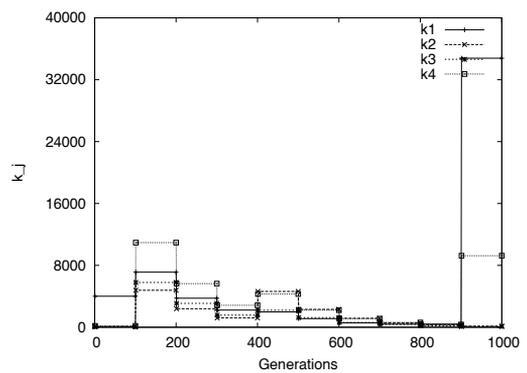


Figure 13. Variant 5 - Test-problem g_{09}

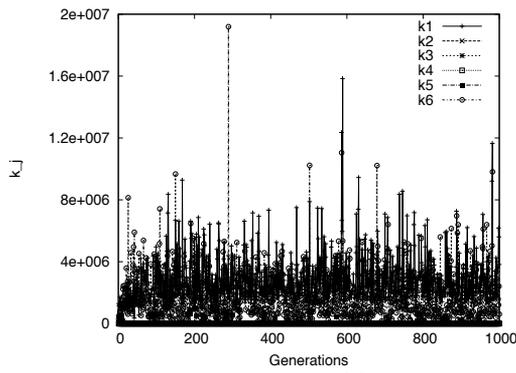


Figure 14. Variant 1 - Test-problem $g04$

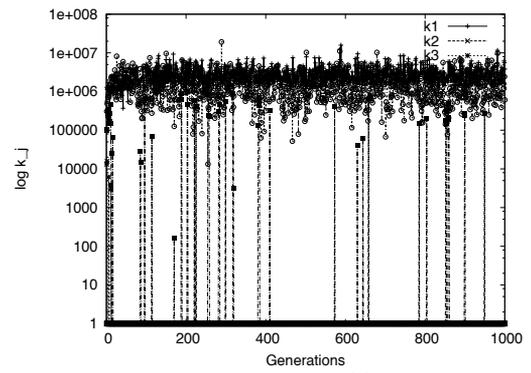


Figure 15. Variant 1 - Test-problem $g04$
(log scale for y -axis)

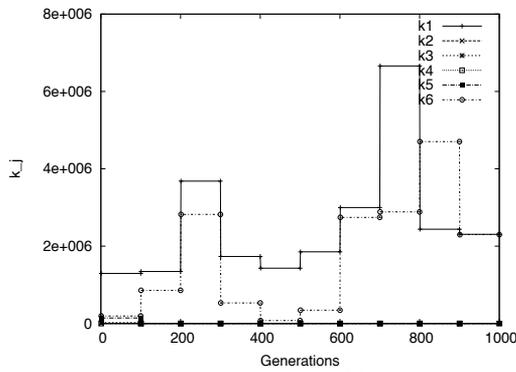


Figure 16. Variant 2 - Test-problem $g04$

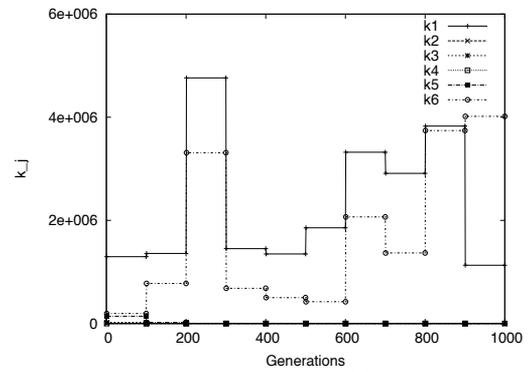


Figure 17. Variant 3 - Test-problem $g04$

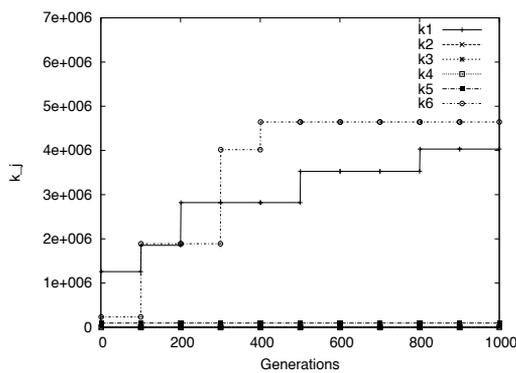


Figure 18. Variant 4 - Test-problem $g04$

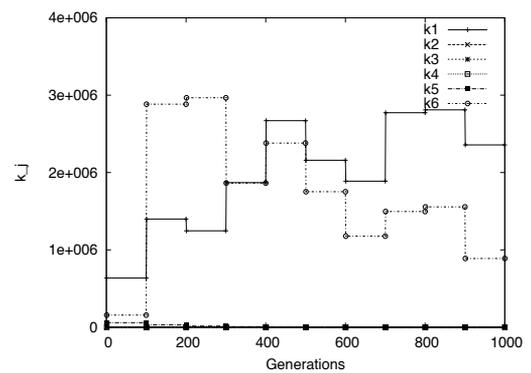


Figure 19. Variant 5 - Test-problem $g04$

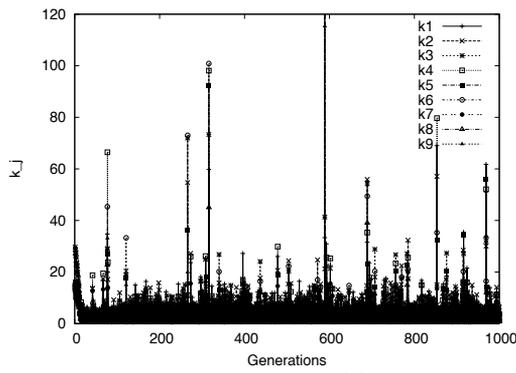


Figure 20. Variant 1 - Test-problem $g01$

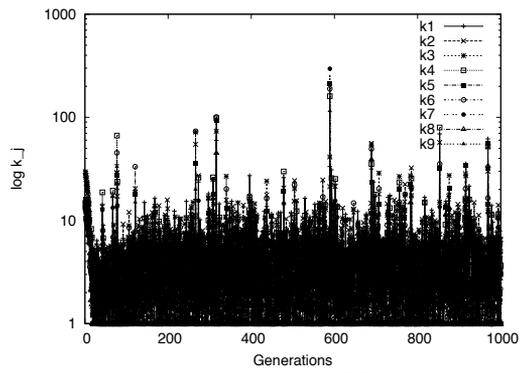


Figure 21. Variant 1 - Test-problem $g01$
(log scale for y -axis)

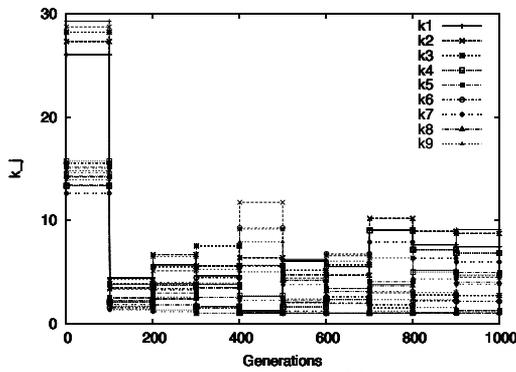


Figure 22. Variant 2 - Test-problem $g01$

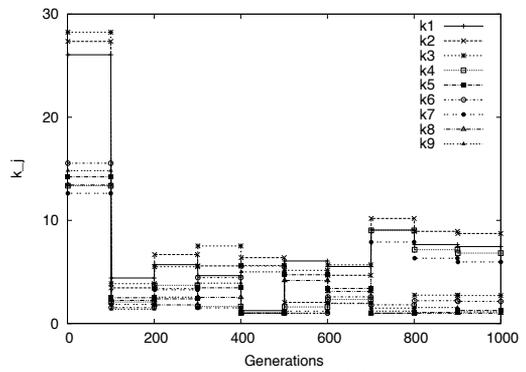


Figure 23. Variant 3 - Test-problem $g01$

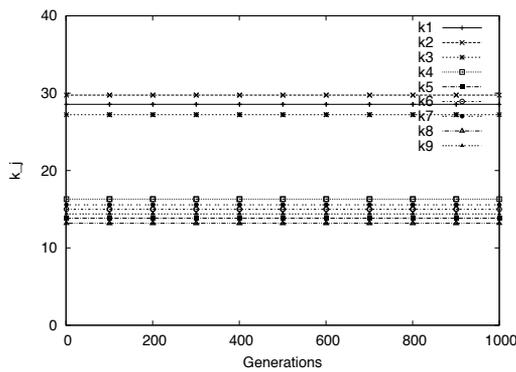


Figure 24. Variant 4 - Test-problem $g01$

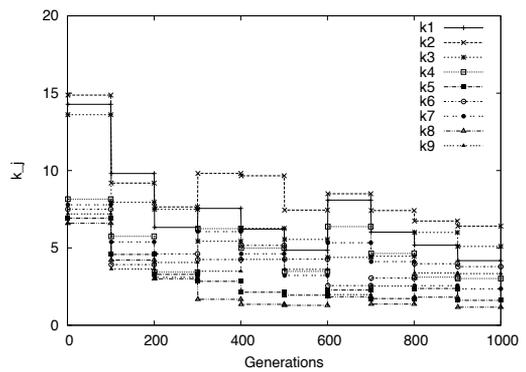


Figure 25. Variant 5 - Test-problem $g01$

5.2 Test 2 - The pressure vessel design

This problem (Sandgren, 1998, Kannan & Kramer, 1995, Deb, 1997, Coello, 2000) corresponds to the weight minimization of a cylindrical pressure vessel with two spherical heads. The objective function involves four variables: the thickness of the pressure vessel (T_s), the thickness of the head (T_h), the inner radius of the vessel (R) and the length of the cylindrical component (L). Since there are two discrete variables (T_s and T_h) and two continuous variables (R and L), one has a nonlinearly constrained mixed discrete-continuous optimization problem. The bounds of the design variables are $0.0625 \leq T_s, T_h \leq 5$ (in constant steps of 0.0625) and $10 \leq R, L \leq 200$. The design variables are given in inches and the weight is written as:

$$W(T_s, T_h, R, L) = 0.6224T_sT_hR + 1.7781T_hR^2 + 3.1661T_s^2L + 19.84T_s^2R$$

to be minimized subject to the constraints

$$\begin{aligned} g_1(T_s, R) &= T_s - 0.0193R \geq 0 & g_2(T_h, R) &= T_h - 0.00954R \geq 0 \\ g_3(R, L) &= \pi R^2 L + 4/3\pi R^3 - 1,296,000 \geq 0 & g_4(L) &= -L + 240 \geq 0 \end{aligned}$$

The first two constraints establish a lower bound to the ratios T_s/R and T_h/R , respectively. The third constraint corresponds to a lower bound for the volume of the vessel and the last one to an upper bound for the length of the cylindrical component.

The Table 3 presents the results of the pressure vessel design using 1000 generations. Two identical results were found for the best value of the final weight (6059.7145293) with respect to the APM variants 2 and 3. Using 5000 generations, the same value of the final weight was found four times as shown in the Table 4. Except for the original APM (variant 1), all variants were able to find the best solutions.

Variant	best	median	Mean	std	worst	frun
1	6059.7151671	6227.8170385	6474.4169921	4.53E+02	7509.8518229	25
2	6059.7145293	6384.0497026	6447.2068164	4.19E+02	7340.3461505	25
3	6059.7145293	6410.2945786	6481.3246679	4.29E+02	7544.4928499	25
4	6059.7187531	6371.1972928	6467.6047851	4.30E+02	7544.5662019	25
5	6059.7151671	6376.8444814	6502.6069921	4.37E+02	7544.4928499	25

Table 3. Results of the pressure vessel design using 1000 generations

variant	best	median	Mean	std	worst	frun
1	6059.7903738	6525.0826625	6571.3568945	4.79E+02	7333.0935211	25
2	6059.7145293	6370.7942228	6448.2773437	4.21E+02	7334.9819158	25
3	6059.7145293	6372.4349034	6445.8280859	4.16E+02	7340.8828459	25
4	6059.7145293	6370.7970442	6466.9971289	4.03E+02	7334.2645178	25
5	6059.7145293	6370.7798337	6445.3129101	4.58E+02	7332.9812503	25

Table 4. Results of the pressure vessel design using 5000 generations

5.3 Test 3 - The cantilever beam design

This test problem (Erbatur et al., 2000) corresponds to the minimization of the volume of the cantilever beam subject to the load P , equal to 50000 N, at its tip. There are 10 design variables corresponding to the height (H_i) and width (B_i) of the rectangular cross-section of each of the five constant steps. The variables B_1 and H_1 are integer, B_2 and B_3 assume discrete values to be chosen from the set $\{2.4, 2.6, 2.8, 3.1\}$, H_2 and H_3 are discrete and chosen from the set $\{45.0, 50.0, 55.0, 60.0\}$ and, finally, B_4 , H_4 , B_5 , and H_5 are continuous. The variables are given in centimeters and the Young's modulus of the material is equal to 200 GPa. The volume of the beam is given by

$$V(H_i, B_i) = 100 \sum_{i=1}^5 H_i B_i$$

subject to:

$$g_i(H_i, B_i) = \sigma_i \leq 14000 \text{ N/cm}^2 \quad i = 1, \dots, 5$$

$$g_{i+5}(H_i, B_i) = H_i/B_i \leq 20 \quad i = 1, \dots, 5$$

$$g_{11}(H_i, B_i) = \delta \leq 2.7 \text{ cm}$$

where δ is the tip deflection of the beam in the vertical direction.

The Table 5 shows the results for the cantilever beam design when 1000 generations were used. The best final volume found (64842.0265521), was achieved by the third variant of the APM variant. Using 5000 generations, the variant 4 provided the best solution among the variants, with a final volume equal to 64578.1955255 as shown in the Table 6.

variant	best	median	mean	std	worst	frun
1	65530.7729044	68752.485343	70236.739375	3.59E+03	78424.35756	25
2	64991.8776321	68990.278258	70642.622188	4.47E+03	84559.93461	25
3	64842.0265521	68662.595134	70312.300781	4.39E+03	78015.93963	25
4	64726.0899470	68924.741220	71383.862812	5.60E+03	85173.95180	25
5	65281.3580659	69142.397989	69860.818906	3.66E+03	78787.491031	25

Table 5. Results of the cantilever beam design using 1000 generations

ariant	best	median	mean	std	worst	frun
1	64578.1959540	68293.437745	67153.214844	2.00E+03	72079.947779	25
2	64578.1957397	64965.027481	66161.232500	1.77E+03	68391.603479	25
3	64578.1959540	64992.041635	66704.774531	3.13E+03	79124.364956	25
4	64578.1955255	68201.363366	66682.130313	1.86E+03	68722.669573	25
5	64578.1957397	65015.052899	66450.535156	1.89E+03	69294.203883	25

Table 6. Results of the cantilever beam design using 5000 generations

5.4 Test 4 - The Tension/Compression Spring design

This example corresponds to the minimization of the volume V of a coil spring under a constant tension/compression load. There are three design variables: the number $x_1 = N$ of active coils of the spring, the winding diameter $x_2 = D$ and the wire diameter $x_3 = d$. The volume of the coil is written as:

$$V(x) = (x_3 + 2)x_2x_1^2$$

and is subject to the constraints:

$$g_1(x) = 1 - \frac{x_2^3x_3}{71875x_1^4} \leq 0 \quad g_2(x) = \frac{x_2(4x_2 - x_1)}{12566x_1^3(x_2 - x_1)} + \frac{2.26}{12566x_1^2} \leq 0$$

$$g_3(x) = 1 - \frac{140.54x_1}{x_2^2x_3} \leq 0 \quad g_4(x) = \frac{x_2 + x_1}{1.5} - 1 \leq 0$$

where

$$0.05 \leq x_1 \leq 0.2 \quad 0.25 \leq x_2 \leq 1.3 \quad 2.0 \leq x_3 \leq 15.0$$

From the Table 7 one can observe that the variants 2, 3 and 4 were able to find the better results for the Tension/Compression Spring design using 1000 generations. The final volume reached is equal to 0.0126973. Using 5000 generations, the previous result for the best volume remains the same (0.0126973) and is again found by the variants 2, 3 and 4 of the APM (Table 8).

Variant	best	median	Mean	std	worst	frun
1	0.0127181	0.0136638	0.0144048	1.82E-03	0.0178499	25
2	0.0126973	0.0135445	0.0144046	1.79E-03	0.0178499	25
3	0.0126973	0.0135445	0.0144074	1.80E-03	0.0178674	25
4	0.0126973	0.0135456	0.0143907	1.78E-03	0.0178499	25
5	0.0127255	0.0140187	0.0146077	1.83E-03	0.0178499	25

Table 7. Results of the coil spring design using 1000 generations

Variant	best	median	mean	std	worst	frun
1	0.0127181	0.0135456	0.0143920	1.82E-03	0.0178499	25
2	0.0126973	0.0135445	0.0143769	1.78E-03	0.0178499	25
3	0.0126973	0.0135445	0.0143706	1.78E-03	0.0178499	25
4	0.0126973	0.0135445	0.0143868	1.80E-03	0.0178499	25
5	0.0127255	0.0127255	0.0146097	1.84E-03	0.0178499	25

Table 8. Results of the coil spring design using 5000 generations

5.5 Test 5 - The 10-bar truss design

This well known test problem (Goldberg & Samtani, 1986), which corresponds to the weight minimization of a 10-bar truss, will be analyzed here subject to constraints involving the stress in each member and the displacements at the nodes. The design variables are the cross-sectional areas of the bars ($A_i, i = 1, 10$). The allowable stress is limited to ± 25 ksi and the displacements are limited to 2 in, in the x and y directions. The density of the material is 0.1 lb/in^3 , Young's modulus is $E = 10^4 \text{ ksi}$, and vertical downward loads of 100 kips are applied at nodes 2 and 4.

Two cases are analyzed: discrete and continuous variables. For the discrete case the values of the cross-sectional areas (in^2) are chosen from the set of 42 options: 1.62, 1.80, 1.99, 2.13, 2.38, 2.62, 2.63, 2.88, 2.93, 3.09, 3.13, 3.38, 3.47, 3.55, 3.63, 3.84, 3.87, 3.88, 4.18, 4.22, 4.49, 4.59, 4.80, 4.97, 5.12, 5.74, 7.22, 7.97, 11.50, 13.50, 13.90, 14.20, 15.50, 16.00, 16.90, 18.80, 19.90, 22.00, 22.90, 26.50, 30.00, 33.50. Using a six-bit string for each design variable, a total of 64 strings are available which are mapped into the 42 allowable cross sections. For the continuous case the minimum cross sectional area is equal to 0.1 in^2 .

For the discrete case all of the variants of the APM were able to find the best result for the final weight of the 10-bar truss structure (5490.738) as detailed in the Table 19. For the continuous case, the original APM (variant 1) found the best solution for the final weight (5062.6605) shown in the Table 10.

variant	best	median	mean	std	worst	frun
1	5490.7378	5599.4444	5605.6312	1.29E+02	6156.3977	25
2	5490.7378	5634.5138	5647.7405	1.94E+02	6487.3525	25
3	5490.7378	5585.0667	5636.5571	2.12E+02	6561.2738	25
4	5490.7378	5628.1253	5669.3382	2.00E+02	6550.6018	25
5	5490.7378	5596.3125	5600.6728	6.38E+01	5719.3412	25

Table 9. Results of the 10-bar truss with respect to the discrete case

variant	best	median	mean	std	worst	frun
1	5062.6605	5088.1406	5116.4183	9.20E+01	5527.0238	25
2	5068.5702	5089.9409	5129.8554	1.20E+03	5631.3084	25
3	5066.8124	5093.6549	5106.9311	5.71E+01	5306.6538	25
4	5068.5005	5091.6265	5138.9569	1.49E+02	5774.8811	25
5	5063.6410	5087.2172	5126.4837	1.37E+02	5705.4190	25

Table 10. Results of the 10-bar truss with respect to the continuous case

5.6 Test 6 - The 120-bar truss

This is a three-dimensional dome structure corresponding to a truss with 120 members (Saka & Ulker, 1991, Ebenau et al., 2005, Capriles et al, 2007).

The dome is subject to a downward vertical equipment loading of 600 kN at its crown (node 1). The displacements are limited to 2 cm, in the x , y and z directions. The density of material is 7860 kg/m^3 and Young's modulus is equal to 21000 kN/cm^2 . The maximum normal stress is limited by $\sigma_{max} = 32.273 \text{ kN/cm}^2$ in tension and compression. Moreover,

the buckling in members under compression is considered, and the stress constraints are given now by:

$$\frac{|\sigma_j|}{\sigma_j^E} - 1 \leq 0, \quad i = 1, 2, \dots, p_\sigma^E \quad \text{with} \quad \sigma_j^E = \frac{kEA_j}{L_j^2}$$

where σ_j is the buckling stress at the j -th member, σ_j^E is the maximum allowable buckling stress for this member, and k is the buckling coefficient, adopted equal to 4. The total number of constraints is equal to 351.

The cross-sectional areas are the sizing design variables and they are to be chosen from a Table containing 64 options (see Capriles et al, 2007).

Variant	best	median	mean	std	worst	frun
1	59.5472	59.9999	60.0266	2.50E-01	60.7749	25
2	59.5472	60.0459	60.2679	6.33E-01	62.3707	25
3	59.7145	60.0459	60.7749	3.05E+00	75.1747	25
4	59.5472	60.0459	60.8720	3.05E+00	75.1747	25
5	59.5472	59.9748	60.0711	4.03E-01	61.0417	25

Table 11. Results of the 120-bar truss

In this experiment the five variants of the APM were able to find the same best solution (59.5472). The fifth variant of the APM provided the best value of the median (59.9748) and the original APM presented the best result corresponding to the mean value.

6. Conclusion

From the experiments performed here one can see that the original APM found more often the best solutions. Among the new variants, the variants 3 and 5 were the most successful. As the performance of variant 4 was slightly inferior to that of variants 3 and 5, one may suspect that the inclusion of the damping procedure was not advantageous.

A more important conclusion may be that there were not large discrepancies among the results obtained by the original APM and the four new variants, which provide computational savings, and that robust variants can be obtained from the overall family of adaptive penalty methods considered here.

7. References

- Barbosa, H.J.C. & Lemonge, A.C.C. (2002). An adaptive penalty scheme in genetic algorithms for constrained optimization problems. *Proceedings of the Genetic and Evolutionary Computation Conference - GECCO*, pp. 287-294, 9-13 July 2002. Morgan Kaufmann Publishers, New York.

- Barbosa, H.J.C. & Lemonge, A.C.C. (2003). A new adaptive penalty scheme for genetic algorithms. *Informations Sciences*, Vol. 156, No. 5, pp. 215–251, 2003.
- Lemonge, A.C.C. & Barbosa, H.J.C. (2004). An adaptive penalty scheme for genetic algorithms in structural optimization. *Int. Journal for Numerical Methods in Engineering*, Vol. 59, No. 5, pp. 703–736, 2004.
- Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
- Goldberg, D.E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Reading, MA, 1989.
- Shoenauer, M. & Michalewicz, Z. (1996). Evolutionary computation at the edge of feasibility. In H-M. Voigt; W. Ebeling; I. Rechenberg and H-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN IV*, Vol. 1141, pp. 245–254, Berlin, 1996. Springer-Verlag, LNCS.
- Koziel, S. & Michalewicz, Z. (1999). Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, Vo. 7, No. 1, pp. 19–44, 1999.
- Liepins, G.E. & Potter, W.D. (1996). A genetic algorithm approach to multiple-fault diagnosis. In L. Davis, editor, *Handbook of Genetic Algorithm*, Chapter 7, pp. 237–250, Boston, USA, 1996. International Thomson Computer Press.
- Orvosh, D. & Davis, L. (1994). A genetic algorithm to optimize problems with feasibility constraints. *Proc. Of the First IEEE Conference on Evolutionary Computation*, pp. 548–553. IEEE Press, 1994.
- Adeli, H. & Cheng, N.-T. (1994). Augmented Lagrangian Genetic Algorithm for Structural Optimization. *Journal of Aerospace Engineering*, Vol. 7, No. 1, pp. 104–118, January 1994.
- Barbosa, H.J.C. (1999) A coevolutionary genetic algorithm for constrained optimization problems. *Proc. of the Congress on Evolutionary Computation*, pp. 1605–1611, Washington, DC, USA, 1999.
- Surry, P.D. & Radcliffe, N.J. (1997). The COMOGA Method: Constrained Optimisation by Multiobjective Genetic Algorithms. *Control and Cybernetics*, Vol. 26, No. 3, 1997.
- Runarsson, T.P. & Yao, X. (2000). Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, Vol. 4, No. 3, pp. 284–294, September 2000.
- Kampen, A.H.C. van, Strom, C.S. & Buydens, L.M.C.(1996). Lethalization, penalty and repair functions for constraint handling in the genetic algorithm methodology. *Chemometrics and Intelligent Laboratory Systems*, Vo. 34, pp. 55–68, 1996.
- Michalewicz, Z. & Shoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, Vol. 4, No. 1, pp. 1–32, 1996.
- Hinterding, R. & Michalewicz, Z. (1998). Your brains and my beauty: Parent matching for constrained optimization. *Proc. of the Fifth Int. Conf. on Evolutionary Computation*, pp. 810–815, Alaska, May 4-9, 1998.
- Koziel, S. & Michalewicz, Z. (1998). A decoder-based evolutionary algorithm for constrained optimization problems, *Proc. of the Fifth Parallel Problem Solving - PPSN*. T. Bäck; A.E. Eiben; M. Shoenauer and H.-P. Schwefel, Editors Amsterdam, September 27-30 1998. Spring Verlag. Lecture Notes in Computer Science.

- Kim, J.-H. & Myung, H. (1997). Evolutionary programming techniques for constrained optimization problems. *IEEE Transactions on Evolutionary Computation*, Vol. 2, No. 1, pp. 129-140, 1997.
- Le Riche, R.G.; Knopf-Lenoir, C. & Haftka, R.T. (1995). A segregated genetic algorithm for constrained structural optimization. *Proc. of the Sixth Int. Conf. on Genetic Algorithms*, pp. 558-565, L.J. Eshelman Editor, Pittsburgh, PA., July 1995.
- Homaifar, H.; Lai, S.H.-Y. & Qi, X. (1994). Constrained optimization via genetic algorithms. *Simulation*, Vol. 62, No. 4, pp. 242-254, 1994.
- Joines, J. & Houck, C. (1994). On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs. *Proc. of the First IEEE Int. Conf. on Evolutionary Computation*. Z. Michalewicz; J.D. Schaffer; H.-P. Schwefel; D.B. Fogel and H. Kitano, Editors, pp. 579-584, June 19-23, 1994.
- Powell, D. & Skolnick, M.M. (1993). Using genetic algorithms in engineering design optimization with non-linear constraints. *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, In S. Forrest, Editor, pp. 424-430, San Mateo, CA, 1993. Morgan Kaufmann.
- Deb, K. (2000) An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, Vol. 186, Nos. 2-4, pp. 311-338, June 2000.
- Michalewicz, Z. (1995). A survey of constraint handling techniques in evolutionary computation. *Proc. of the 4th Int. Conf. on Evolutionary Programming*, pp. 135-155, Cambridge, MA, 1995. MIT Press.
- Michalewicz, Z.; Dasgupta, D; Le Riche, R.G. & Shoenauer, M. (1996). Evolutionary algorithms for constrained engineering problems. *Computers & Industrial Engineering Journal*, Vol. 30, No. 2, pp. 851-870, 1996.
- Bean, J.C. & Alouane, A.B. (1992). A dual genetic algorithm for bounded integer programs. *Technical Report TR 92-53*, Department of Industrial and Operations Engineering, The University of Michigan, 1992.
- Shoenauer, M. & Xanthakis, S. (1993). Constrained GA optimization. *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, S. Forrest, editor, pp. 573-580, Los Altos, CA, 1993. Morgan Kaufmann Publishers.
- Coit, D.W.; Smith, A.E. & Tate, D.M. (1996). Adaptive penalty methods for genetic optimization of constrained combinatorial problems. *INFORMS Journal on Computing*, Vo. 6, No. 2, pp. 173-182, 1996.
- Yokota, T ; Gen, M. ; Ida, K. & Taguchi, T. (1995). Optimal design of system reliability by an improved genetic algorithms. *Trans. Inst. Electron. Inf. Comput. Engrg.*, J78-A(6), pp. 702-709, 1995.
- Gen, M. & Gheng, R. (1996a). Interval programming using genetic algorithms. *Proc. of the Sixth International Symposium on Robotics and Manufacturing*, Montepelcier, France, 1996.
- Gen, M. & Gheng, R. (1996b). A survey of penalty techniques in genetic algorithms. *Proc. of the 1996 International Conference on Evolutionary Computation*, IEEE, pp. 804-809, Nagoya, Japan, UK, 1996.
- Hamida, S.B. & Shoenauer, M. (2000). An adaptive algorithm for constrained optimization problems. *Parallel Problem Solving from Nature - PPSN VI*, Vo. 1917, pp. 529-538, Berlin. Springer-Verlag. Lecture Notes in Computer Science, 2000.

- Zavislak, M.J. (2004). Constraint handling in groundwater remediation design with genetic algorithms, 2004. *M.Sc. Thesis*, Environmental and Civil Engineering, The Graduate College of the Illinois University at Urbana-Champaign.
- Gallet, C. ; Salaun, M. & Bouchet, E. (2005). An example of global structural optimisation with genetic algorithms in the aerospace field. *VIII International Conference on Computational Plasticity - COMPLAS VIII*, CIMNE, Barcelona, 2005.
- Obadage, A.S. & Hampornchai, N. (2006). Determination of point of maximum likelihood in failure domain using genetic algorithms. *International Journal of Pressure Vessels and Piping*, Vol. 83, No. 4, pp. 276–282, 2006.
- Sandgren, E. (1998). Nonlinear integer and discrete programming in mechanical design. *Proc. of the ASME Design Technology Conference*, pp. 95–105, Kissimee, FL, 1988.
- Kannan, B.K. & Kramer, S.N. (1995). An augmented Lagrange multiplier based method for mixed integer discrete continuous optimization and its applications to mechanical design. *Journal of Mechanical Design*, Vol. 116, pp. 405–411, 1995.
- Deb, K. (1997). GeneAS: A robust optimal design technique for mechanical component design. *Evolutionary Algorithms in Engineering Applications*, pp. 497–514. Springer-Verlag, Berlin, 1997.
- Coello, C.A.C. (2000). Use of a Self-Adaptive Penalty Approach for Engineering Optimization Problems. *Computers in Industry*, Vol. 41, No. 2, pp. 113–127, January 2000.
- Erbatur, F. ; Hasançebi, I. ; Tütüncü, I. & Kilç, H. (2000). Optimal design of planar and space structures with genetic algorithms. *Computers & Structures*, Vo. 75, pp. 209–224, 2000.
- Goldberg, D.E. & Samtani, M.P. (1986). Engineering optimization via genetic algorithms. *Proc. 9th Conf. Electronic Computation*, ASCE, pp. 471–482, New York, NY, 1986.
- Saka, M.P. & Ulker, M. (1991). Optimum design of geometrically non-linear space trusses. *Computers and Structures*, Vol. 41, pp. 1387–1396, 1991.
- Ebenau, C. ; Rotsschafer, J. & Thierauf, G. (2005). An advance evolutionary strategy with an adaptive penalty function for mixed-discrete structural optimisation. *Advances in Engineering Software*, Vo. 36, pp. 29–38, 2005.
- Capriles, P.V.S.Z.; Fonseca, L.G. ; Barbosa, H.J.C & Lemonge, A.C.C. (2007). Rank-based ant colony algorithms for truss weight minimization with discrete variables. *Communications in Numerical Methods in Engineering*, Vol. 26, No. 6, pp. 553–576, 2007.

Evolutionary-Based Control Approaches for Multirobot Systems

Jekanthan Thangavelautham, Timothy D. Barfoot and
Gabriele M.T. D'Eleuterio
*Institute for Aerospace Studies, University of Toronto
Canada*

1. Introduction

In this chapter, we summarize and synthesize our investigations of the use of evolutionary algorithms to automatically program robots, particularly for application to space exploration. In the Space Robotics Group at the University of Toronto Institute for Aerospace Studies, we were motivated to begin work in this area a decade ago when the concept of *network science* became popular in the space exploration community. Network science commonly refers to science that requires a distribution of possibly simultaneous measurement devices or a distribution of platforms on a planetary body. Consider, for example, seismology studies of an alien body that will require sending a signal from one point on the surface to be read at several other points in order to analyze the material characteristics of the body. Or, consider the development of a very-low frequency array (VLFA) on the Moon to allow for heretofore unattainable astrophysical observations using radio astronomy. Such an observatory will require a number of dipole units deployed over a region of a few hundred square kilometres.

Our original thoughts were that these and other network science experiments could be implemented using a network of small mobile robots, similar to a colony of ants. It is possible for millions of ants to act as a superorganism through local pheromone communication. Thomas (1974) perhaps describes this phenomenon best:

A solitary ant, afield, cannot be considered to have much of anything on his mind. Four ants together, or ten, encircling a dead moth on a path, begin to look more like an idea. But it is only when you watch the dense mass of thousands of ants, blackening the ground that you begin to see the whole beast, and now you observe it thinking, planning, calculating. It is an intelligence, a kind of live computer, with crawling bits for its wits.

We set out to reproduce this type of behaviour in a *multirobot system* (i.e., a network of mobile robots) for application to space exploration.

As we began investigating how to devise control schemes for the network science tasks, additional applications came to mind including deployment of solar cell arrays on a planetary surface, site preparation for a lunar base, and gathering objects of interest for analysis or in-situ resource utilization. Although these additional tasks did not necessitate

the use of a group of robots, there are certain advantages offered by this choice. Redundancy and fault tolerance are fundamental attributes of any reliable space system. By using a group of robots, we might afford to lose a small number of individuals and yet still accomplish our desired task. The flip side to redundancy is taking risks. By making the system modular and thus redundant, we could be willing to accept more risk in the design of a single robot because it no longer has the potential to produce a single point failure.

In many of our early experiments, we tried designing controllers for our groups of robots by hand (Earon et al., 2001). This was possible for some simple tasks, such as having the robots position themselves in a particular geometric formation. As we became interested in the resource collection and array deployment tasks, the burden of manual programming became higher and we turned to the use of evolutionary algorithms.

Moreover, we wondered if it would be possible to specify the required task at the group level and have the evolutionary algorithm find the best way to coordinate the robots to accomplish the overall goal. This notion of top-down performance specification is very much in keeping with the formal approach to space engineering, in which mission-level goals are provided and then broken down to manageable pieces by a designer. Accordingly, this notion of *task decomposition* is at the heart of our discussion throughout this chapter. We will advocate for an approach that does not explicitly break a task down into subtasks for individual robots, but rather facilitates this through careful selection of the evaluation criteria used to gauge group behaviour on a particular task (i.e., the *fitness function*). By using an evolutionary algorithm with this judiciously chosen fitness function, task decomposition occurs through *emergence* (self-organization).

The remainder of this chapter is organized as follows. First, we review the literature on the use of evolutionary algorithms for task decomposition and development of multirobot controllers. Next we report on a number of approaches we have investigated to control and coordinate groups of robots. Our discussion is framed in the context of four tasks motivated by space exploration: *heap formation* (Barfoot & D'Eleuterio, 2005), *tiling pattern formation* (Thangavelautham & D'Eleuterio, 2004), *a walking robot* (Barfoot et al., 2006) (wherein each leg can be thought of a single robot), and *resource gathering* (Thangavelautham et al., 2007). This is followed by a discussion of the common findings across these experiments and finally we make some concluding remarks.

2. Background

Task decomposition involves partitioning/segmenting of a complex task into subtasks. The partitioning is expected to facilitate solving the simpler subtasks which in turn are combined to yield a solution to the overall task. One approach to role assignment and execution of the subtasks is through use of multirobot systems. Multirobot systems offer the security of redundancy, fault tolerance and, depending on the task, scalability. They furthermore allow for the parallelization of operations. With the use of multiple agents or robots, their control and coordination are critical.

In nature, multiagent systems such as social insects use a number of mechanisms for control and coordination. These include the use of *templates*, *stigmergy*, and *self-organization*. *Templates* are environmental features perceptible to the individuals within the collective (Bonabeau et al., 1999). *Stigmergy* is a form of indirect communication mediated through the environment (Grassé, 1959). In insect colonies, templates may be a natural phenomenon or they may be created by the colonies themselves. They may include temperature,

humidity, chemical, or light gradients. In the natural world, one way in which ants and termites exploit stigmergy is through the use of pheromone trails. *Self-organization* describes how local or microscopic behaviours give rise to a macroscopic structure in systems (Bonabeau et al., 1997). However, many existing approaches suffer from another emergent feature called *antagonism* (Chantemargue et al., 1996). This describes the effect that arises when multiple agents that are trying to perform the same task interfere and reduce the overall efficiency of the group.

Within the field of robotics, many have sought to develop multirobot control and coordination behaviours based on one or more of the prescribed mechanisms used in nature. These solutions have been developed using user-defined deterministic ‘if-then’ or preprogrammed stochastic behaviours. Such techniques in robotics include template-based approaches that exploit light fields to direct the creation of circular walls (Stewart and Russell, 2003), linear walls (Wawerla et al., 2002) and planar annulus structures (Wilson et al., 2004). Stigmergy has been used extensively in collective-robotic construction tasks, including blind bull dozing (Parker et al., 2003), box pushing (Matarić et al., 1995) and heap formation (Beckers et al., 1994).

Inspired by insect societies the robot controllers are often designed to be reactive and have access only to local information. They are nevertheless able to self-organize, through cooperation to achieve an overall objective. This is difficult to do by hand, since the global effect of these local interactions is often hard to predict. The simplest hand-coded techniques have involved designing a controller for a single robot and scaling to multiple units by treating other units as obstacles to be avoided (Parker et al., 2003) (Stewart & Russell, 2003), (Beckers et al., 1994). Other more sophisticated techniques involve use of explicit communication or designing an extra set of coordination rules to handle graceful agent-to-agent interactions (Wawerla et al., 2002). These approaches are largely heuristic and rely on ad hoc assumptions that often require knowledge of the task domain.

In contrast, machine learning techniques (particularly artificial evolution) exploit self-organization and relieve the designer of the need to determine a suitable control strategy. The controllers in turn are designed from the start with cooperation and interaction, as a product of emergent interactions with the environment. It is more difficult to design controllers by hand with cooperation in mind because it is difficult to predict or control the global behaviours that will result from local interactions. Designing successful controllers by hand can devolve into a process of trial and error.

A means of reducing the effort required in designing controllers by hand is to encode controllers as Cellular Automata (CA) look-up tables and allow a genetic algorithm to evolve the table entries (Das et al., 1995). The assumption is that each combination of discretized sensory inputs will result in an independent choice of discretized output behaviours. This approach is an instance of a ‘tabula rasa’ technique, whereby a control system starts off with a blank slate with limited assumptions regarding control architecture and is guided through training by a fitness function (system goal function).

As we show in this chapter, this approach is used successfully to solve a multiagent heap formation task (Barfoot & D’Eleuterio, 1999) and a 2×2 tiling formation task (Thangavelautham et al., 2003). Robust decentralized controllers that exploit stigmergy and self-organization are found to be scalable to ‘world size’ and to agent density. Look-up table approaches are also beneficial for hardware experiments where there is minimal computational overhead incurred as a result of sensory processing.

We also wish to analyze the scalability of evolutionary techniques to bigger problem spaces. One of the limitations with a look-up table approach is that the table size grows exponentially to the number of inputs. For the 3×3 tiling formation task, a monolithic look-up table architecture is found to be intractable due to premature search stagnation. To address this limitation, the controller is modularized into 'subsystems' that have the ability to explicitly communicate and coordinate actions with other agents (Thangavelautham et al., 2003). This act of dividing the agent functionality into subsystems is a form of user-assisted task decomposition through modularization. Although the technique uses a global fitness function, such design intervention requires domain knowledge of the task and ad hoc design choices to facilitate searching for a solution.

Alternatively, CA lookup tables could be networked to exploit inherent modularity in a physical system during evolution, such as series of locally coupled leg controllers for a hexapod robot (Earon et al., 2000). This is in contrast to some predefined recurrent neural network solutions such as by (Beer & Gallagher, 1992), (Parker & Li, 2003) that are used to evolve 'leg cycles' and gait coordination in two separate stages. This act of performing staged evolution involves a human supervisor decomposing a walking gait task between local cyclic leg activity and global, gait coordination. In addition, use of recurrent neural networks for walking gaits requires fairly heavy online computations to be performed in real time, in contrast to the much simpler network of CA lookup tables.

Use of neural networks is also another form of modularization, where each neuron can communicate, perform some form of sensory information processing and can acquire specialized functionality through training. The added advantage of neural network architectures is that the neurons can also generalize unlike a CA lookup table architecture, by exploiting correlations between a combination of sensory inputs thus effectively shrinking the search space. Fixed topology neural networks architectures have been extensively used for multirobot tasks, including building walls, corridors and briar patches (Crabbe & Dyer, 1999) and cooperative transport (Groß & Dorigo, 2003).

However, fixed topology monolithic neural network architectures are also faced with scalability issues. With increased numbers of hidden neurons, one is faced with the effects of *spatial crosstalk* where noisy neurons interfere and drown out signals from feature-detecting neurons (Jacob et al., 1991). Crosstalk in combination with limited supervision (through use of a global fitness function) can lead to the 'bootstrap problem' (Nolfi & Floreano, 2000), where evolutionary algorithms are unable to pick out incrementally better solutions for crossover and mutation resulting in premature stagnation of the evolutionary run. Thus, choosing the wrong network topology may lead to a situation that is either unable to solve the problem or difficult to train (Thangavelautham & D'Eleuterio, 2005).

A critical element of applying neural networks to robotic tasks is how best to design and organize the neural network architecture to facilitate self-organized task decomposition and overcome the 'bootstrap problem'. For these tasks, we may use a global fitness function that doesn't explicitly bias towards a particular task decomposition strategy.

For example, the tiling formation task could be arbitrarily divided into a number of subtasks, including foraging for objects, redistributing object piles, arranging objects in the desired tiling structure locally, merging local lattice structures, reaching a collective consensus and finding/correcting mistakes in the lattice structure. Instead, with less supervision, we rely on the robot controller themselves to determine how best to decompose and solve the task through an artificial Darwinian process.

This is in contrast to other task decomposition techniques that require more supervision including *shaping* (Dorigo & Colombetti, 1998) and *layered learning* (Stone & Veloso, 2000). *Shaping* involves controllers learning on a simplified task with the task difficulty being progressively increased through modification of learning function until a desired set of behaviours emerge. *Layered learning* involves a supervisor partitioning a task into a set of simpler goal functions (corresponding to subtasks). These subtasks are learned sequentially until the controller can solve the corresponding task. Both of these traditional task decomposition strategies rely on supervisor intervention and domain knowledge of a task at hand. For multirobot applications, the necessary local and global behaviours need to be known *a priori* to make decomposition steps meaningful. We believe that for a multirobotic system, it is often easier to identify and quantify the system goal, but determining the necessary cooperative behaviours is often counterintuitive. Limiting the need for supervision also provides numerous advantages including the ability to discover novel solutions that would otherwise be overlooked by a human supervisor.

Fixed-topology ensemble network architectures such as the Mixture of Experts (Jacob et al., 1991), Emergent Modular architecture (Nolfi, 1997) and Binary Relative Lookup (Thangavelautham & D'Eleuterio, 2004) in evolutionary robotics use a gating mechanism to preprocess sensor input and assign modular 'expert' networks to handle specific subtasks. Assigning expert networks to handle aspects of a task is a form of task decomposition. Ensemble networks consist of a hierarchical modularization scheme where networks of neurons are modularized into experts and the gating mechanism used to arbitrate and perform selection amongst the experts. Mixture of Experts uses pre-assigned gating functions that facilitate cooperation amongst the 'expert networks' while Nolfi's emergent modular architecture uses gating neurons to select between two output neurons. The BRL architecture is less constrained, as both the gating mechanism and expert networks are evolved simultaneously and it is scalable to a large number of expert networks.

The limitation with fixed-topology ensemble architectures is the need for supervisor intervention in determining the required topology and number of expert networks. In contrast, with variable length topologies, the intention is to evolve both network architecture and neuronal weights simultaneously. Variable length topologies such as Neuro-Evolution of Augmenting Topologies (NEAT) (Stanley & Miikkulainen, 2002) use a one-to-one mapping from genotype to the phenotype. Other techniques use recursive rewriting of the genotype contents to produce a phenotype such as Cellular Encoding (Gruau, 1994), L-systems (Sims, 1994), Matrix Rewriting (Kitano, 1990), or exploit artificial *ontogeny* (Dellaert & Beer, 1994). *Ontogeny* (morphogenesis) models developmental biology and includes a growth program in the genome that starts from a single egg and subdivides into specialized daughter cells. Other morphogenetic systems include (Bongard & Pfeifer, 2001) and Developmental Embryonal Stages (DES) (Federici & Downing, 2006).

The growth program within many of these morphogenetic systems is controlled through artificial gene regulation. Artificial gene regulation is a process in which gene activation/inhibition regulate (and is regulated by) the expression of other genes. Once the growth program has been completed, there is no further use for gene regulation within the artificial system, which is in stark contrast to biological systems where gene regulation is

always present. In addition, these architectures lack any explicit mechanism to facilitate network modularization evident with the ensemble approaches and are merely variable representations of standard neural network architectures. These variable-length topologies also have to be grown incrementally starting from a single cell in order to minimize the dimensional search space since the size of the network architecture may inadvertently make training difficult (Stanley & Miikkulainen, 2001). With recursive rewriting of the phenotype, limited mutations can result in substantial changes to the growth program. Such techniques also introduce a *deceptive fitness landscape* where limited fitness sampling of a phenotype may not correspond well to the genotype resulting in premature search stagnation (Roggen & Federici, 2004).

Artificial Neural Tissues (Thangavelautham and D'Eleuterio, 2005) address limitations evident with existing variable length topologies through modelling of a number of biologically-plausible mechanisms. Artificial Neural Tissues (ANT) includes a coarse-coding-based neural regulatory system that is similar to the network modularity evident in fixed-topology ensemble approaches. ANT also uses a nonrecursive genotype-to-phenotype mapping avoiding deceptive fitness landscapes, and includes gene duplication similar to DES. Gene duplication involves making redundant copies of a master gene and facilitates *neutral complexification*, where the copied gene undergoes mutational drift and results in expression of incremental innovations (Federici & Downing, 2006). In addition, both gene and neural-regulatory functionality limits the need to grow the architecture incrementally, as there exist mechanisms to selectively activate and inhibit parts of a tissue even after completion of the growth program.

A review of past work highlights the possibility of training multirobot controllers with limited supervision using only a global fitness function to perform self-organized task decomposition. These techniques also show that by exploiting hierarchical modularity and regulatory functionality, controllers can overcome tractability concerns. In the following sections, we explore a number of techniques we have used in greater detail.

3. Tasks

3.1 Heap-Formation

The heap-formation task or object-clustering has been extensively studied and is analogous to the behaviour in some social insects (Deneubourg et al., 1991). In the space exploration context, this is relevant to gathering rocks or other materials of interest. It is believed that this task requires global coordination for a group of decentralized agents, existing in a two-dimensional space, to move some randomly distributed objects into a single large cluster (Fig. 1). Owing to the distributed nature of the agents, there is no central controlling agent to determine where to put the cluster and the agents must come to a common decision among themselves without any external supervision (analogous to the global partitioning task in cellular automata work (Mitchell et al., 1996)). The use of distributed, homogenous sets of agents exploits both redundancy and parallelism. Each agent within the collective has limited sensory range and lacks a global blueprint of the task at hand but cooperative coordination amongst agents can, as we show here, make up for these limitations (Barfoot & D'Eleuterio, 1999); (Barfoot & D'Eleuterio, 2005).

To make use of Evolutionary Algorithms (EAs), a fitness function needs to be defined for the task. Herein we define a fitness function that can facilitate selection of controllers for the task at hand without explicitly biasing for a particular task decomposition strategy or set of

behaviours. In contrast to this idea, the task could be manually decomposed into a number of potential subtasks, including foraging for objects, piling objects found into small transitional piles, merging small piles into larger ones and reaching a collective consensus in site selection for merging all the piles. Traditional fitness functions such as (Dorigo & Colombetti, 1998) involve summing separate behaviour shaping functions that explicitly tune the controllers towards predetermined set of desired behaviours. With multiagent systems, it is not always evident how best to systematically determine these behaviours. It is often easier to identify the global goals within the system but not the coordination behaviours necessary to accomplish the global goals. Thus, the fitness functions we present here perform an overall global fitness measure of the system and lack explicit shaping for a particular set of behaviours. The intention is for the multiagent system to self-organize into cooperatively solving the task.

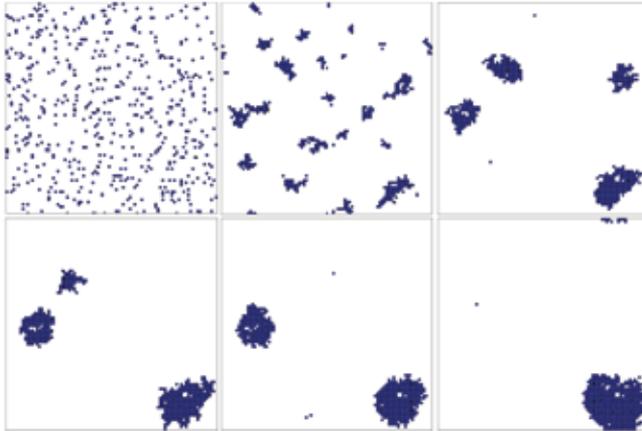


Figure 1. Typical snapshots of system at various time steps (0; 1010; 6778; 14924; 20153; 58006). The world size is 91×90 ; there are 270 agents and 540 objects. Only the objects (dark circles) are shown for clarity

For the heap formation task, the two dimensional grid world in which the agents exist is broken into J bins, A_j , of size $l \times l$. We use a fitness function based on Shannon's entropy as defined below:

$$f_i = 1 + \frac{\sum_{j=1}^J q_j \ln q_j}{\ln J} \quad (1)$$

where q_j is defined as follows:

$$q_j = \frac{n(A_j)}{\sum_{j=1}^J n(A_j)} \quad (2)$$

$n(A_j)$ is the number of objects in bin A_j so that $0 \leq f_i \leq 1$. To summarize, fitness is assigned to a controller by equipping each agent in a collective with the same controller. The collective is allowed to roam around in a two-dimensional space that has a random initial distribution of objects. At the end of T time-steps, f_i is calculated, which indicates how well the objects are clustered. This is all repeated I times (varying initial conditions) to determine the average fitness.

3.1.1 Cellular Automata

To perform the task, each robot-like agent is equipped with a number of sensors and actuators. To relate this to real robots, it will be assumed that some transformation may be performed on raw sensor data so as to achieve a set of orthogonal (Kube & Zhang, 1996) virtual sensors that output a discrete value. This transformation is essentially a preprocessing step that reduces the raw sensor data to more readily usable discretized inputs. Let us further assume that the output of our control system may be discrete. This may be done by way of a set of *basis behaviours* (Matarić, 1997). Rather than specify the actuator positions (or velocities), we assume that we may select a simple behaviour from a finite predefined palette. This may be considered a post-processing step that takes a discretized output and converts it to the actual actuator control. The actual construction of these transformations requires careful consideration but is also somewhat arbitrary.

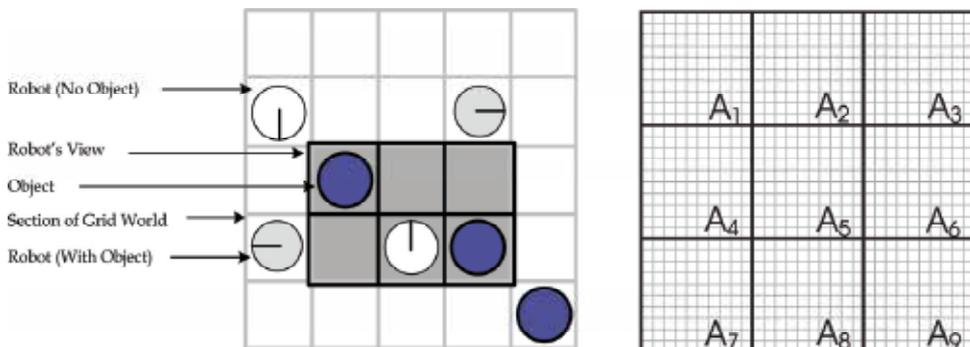


Figure 2. (Left) Typical view of a simulated robot. Circles (with the line indicating orientation) are robots. Dark circles are objects. (Right) Partition of grid world into bins for fitness calculation

Once the pre/post-processing has been set up, the challenge remains to find an appropriate arbitration scheme that takes in a discrete input sequence (size N) and outputs the appropriate discrete output (one of M basis behaviours). The simulated robot's sensor input layout is shown in Fig. 2 (left). The number of possible views for the robot is $3^5 \times 2 = 486$. Each of 5 cells can be free, occupied by an object or by another robot. The robot itself can either be holding an object or not. For an arbitration scheme, we use a lookup-table similar to Cellular Automata, in which the axes are the sensory inputs and the contents of the table, the output behaviours. It could be argued that CAs are the simplest example of a multiagent system for the heap formation task. With 2 basis behaviours and a CA lookup-table of size 486, there are 2^{486} possible CA lookup-tables. This number is quite large but, as we will see, good solutions can still be found. It should be pointed out that our agents will be functioning in a completely deterministic manner. From a particular initial condition, the system will always unfold in the same particular way.

3.1.2 Experiments

Fig. 3 (left) shows a typical evolutionary run using CA lookup table controllers. Analysis of the population best taken after 150 generations shows that controllers learn to form small piles of objects, which are over time merged into a single large pile. Even with a very simple controller, we can obtain coordinated behaviour amongst the agents. The agents

communicate indirectly among themselves through stigmergy (by manipulating the environment). This is primarily used to seed piles that are in turn merged into clusters. The benefits of using a decentralized multiagent controller is that the system may also be rescaled to a larger world size.

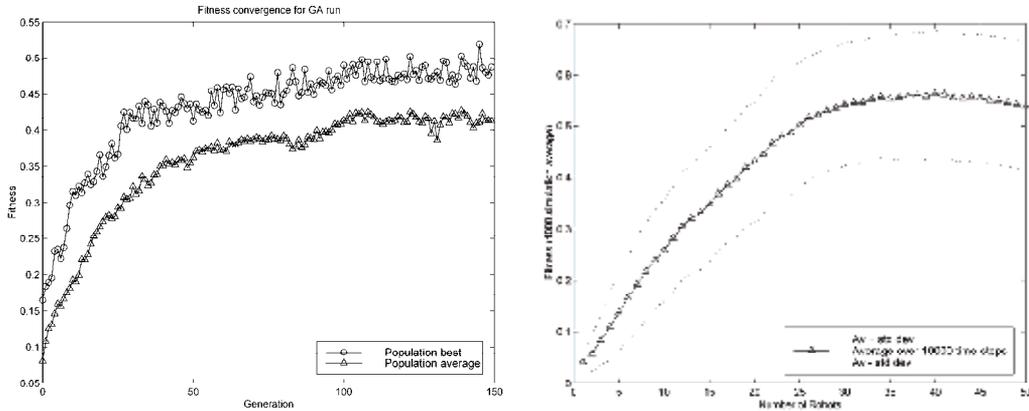


Figure 3. (Left) Convergence history of a typical EA run with a population size of 50. (Right) Max system fitness with number of robots rescaled. Solution was evolved with 30 simulated robots

The controllers were evolved with one particular set of parameters (30 robots, 31×30 world size, 60 objects) but the densities of agents and resources can be rescaled, without rendering the controllers obsolete. This particular trait gives us a better understanding of the robustness of these systems and some of their limitations. Altering the agent density, while keeping all other parameters constant, shows the system performs best under densities slightly higher than during training as shown in Fig. 3 (right), accounting for a few agents getting stuck. With too few agents, the system is under-populated and hence takes longer for coordination while too many agents result in disrupting the progress of the system due to *antagonism*. Thus maintaining a constant density scaling with respect to the training parameters, the overall performance of the system compares well when scaled to larger worlds. What we witness from these experiments is that with very simple evolved multiagent controllers, it is feasible to rescale the system to larger world sizes.

3.2 Tiling Pattern Formation

The tiling pattern formation task (Thangavelautham et al., 2003), in contrast to the heap-formation task, involves redistributing objects (blocks) piled up in a two-dimensional world into a desired tiling structure (Fig. 4). In a decentralized setup, the agents need to come to a consensus and form one 'perfect' tiling pattern. This task also draws inspiration from biology, namely a termite-nest construction task that involves redistributing pheromone-filled pellets on the nest floor (Deneubourg, 1977). Once the pheromone pellets are uniformly distributed, termites use the pellets as markers for constructing pillars to support the nest roof.

In contrast to our emergent task decomposition approach, the tiling pattern formation task could be arbitrarily decomposed into a number of potential subtasks. These may include foraging for objects (blocks), redistributing block piles, arranging blocks in the desired tiling

structure locally, merging local lattice structures, reaching a collective consensus and finding/correcting mistakes in the lattice structure. Instead, we are interested in evolving homogenous decentralized controllers (similar to a nest of termites) for the task without need for human assisted task decomposition.

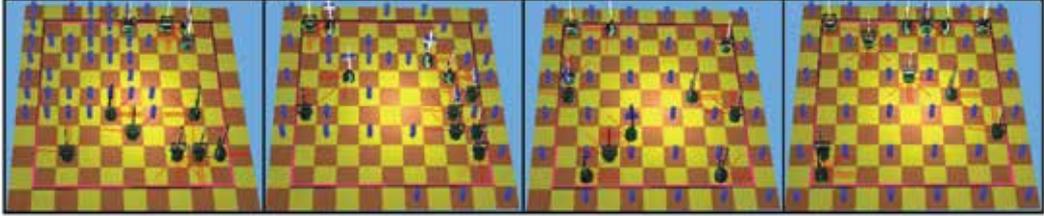


Figure 4. Typical simulation snapshots at various timesteps (0; 100; 400; 410) taken for the 2×2 tiling formation task. Solutions were evolved on an 11×11 world (11 robots, 36 blocks)

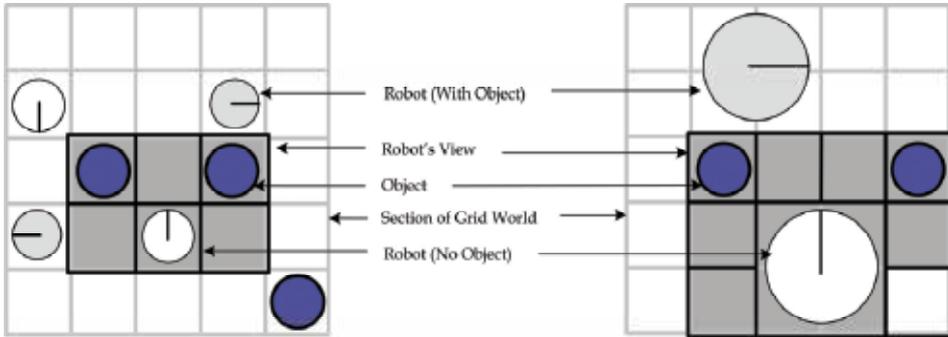


Figure 5. Typical view of a simulated robot for the 2×2 (left) and 3×3 (right) tiling formation tasks. Each robot can sense objects, other agents and empty space in the 5 (left) and 7 (right) shaded squares surrounding the robot

As shown earlier with the heap formation task, decentralized control offers some inherent advantages including the ability to scale up to a larger problem size. Furthermore, task complexity is dependent on the intended tile spacing, because more sensors would be required to construct a 'wider' tiling pattern. In addition, we use Shannon's entropy to be a suitable fitness function for the tiling formation task. For the $m \times m$ tiling pattern formation task, we use Eq. 3 as the fitness function, with q_j used from Eq. 2.

$$f_i = \frac{-\sum_{j=1}^J q_j \ln q_j}{\ln J} \quad (3)$$

The sensor input layouts for the simulated robots used for the 2×2 and 3×3 tiling formation task are shown in Fig. 5.

3.2.1 Emergent Task-Decomposition Architectures

It turns out that the 3×3 tiling pattern formation task is computationally intractable for EAs to find a viable monolithic CA lookup table. To overcome this hurdle, we also considered the use of neural networks as multiagent controllers. Here we discuss a modular neural network architecture called Emergent Task-Decomposition Networks (ETDNs). ETDNs

(Thangavelautham et al., 2004) consist of a set of decision networks that mediate competition and a modular set of expert network that compete for behaviour control. The role of decision networks is to preprocess the sensory input and explicitly ‘select’ for a specialist expert network to perform an output behaviour. A simple example of an ETDN architecture includes a single decision neuron arbitrating between two expert networks as shown in Fig. 6. This approach is a form of task decomposition, whereby separate expert modules are assigned handling of subtasks, based on an evolved sensory input-based decision scheme.

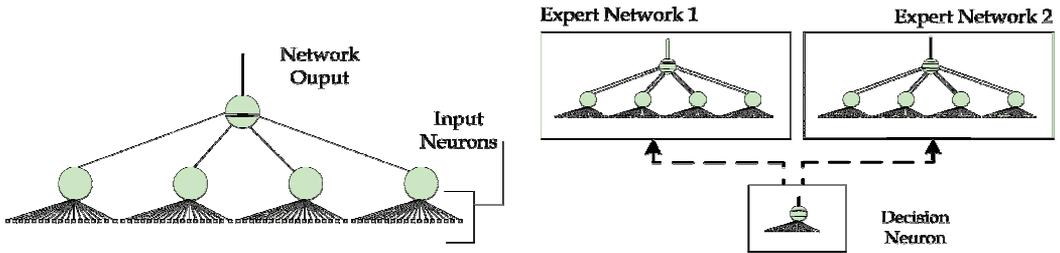


Figure 6. (Left) An example of the non-emergent network used in our experiments. (Right) ETDN architecture consisting of a decision neuron that arbitrates between 2 expert networks

The architecture exploits network modularity, evolutionary competition and specialization to facilitate emergent (self-organized) task decomposition. Unlike traditional machine learning methods, where handcrafted learning functions are used to train the decision and expert networks separately, ETDN architectures require only a global fitness function. The intent is for the architecture to evolve the ability to decompose a complex task into a set of simpler tasks with limited supervision.

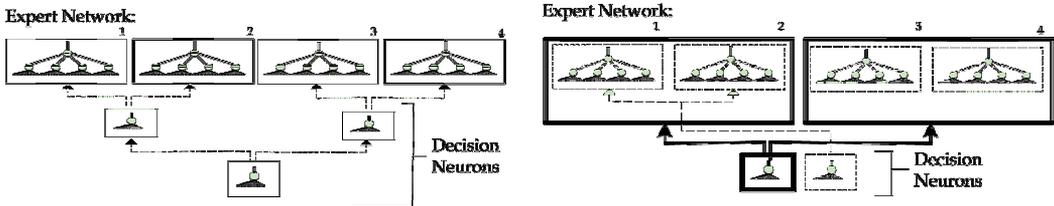


Figure 7. (Left) BDT Architecture with 4 expert networks and 3 decision neurons. (Right) BRL Architecture with 4 expert networks and 2 decision neurons

The ETDNs can also be generalized for n_E expert networks. Here we discuss two extensions to the ETDN architecture, namely the Binary Relative Lookup (BRL) architecture and Binary Decision Tree (BDT) architecture (see Fig. 7). The Binary Relative Lookup (BRL) architecture consists of a set of n_D unconnected decision neurons that arbitrate among 2^{n_D} expert networks. Starting from left to right, each additional decision neuron determines the specific grouping of expert networks relative to the selected group. Since the decision neurons are unconnected, this architecture is well-suited for parallel implementation.

The Binary Decision Tree (BDT) architecture is represented as a binary tree where the tree nodes consist of decision neurons and the leaves consist of expert networks. For this architecture, n_D decision neurons arbitrate among $n_D + 1$ expert networks. The tree is traversed by starting from the root and computing decision neurons along each selected branch node until an expert network is selected. Unlike BRLs, there is a one-to-one

mapping between the set of decision neurons output states and the corresponding expert network. The computational cost of the decision neurons for both architectures is $C_D \propto \log n_E$.

We also introduce modularity within each neuron, through the use of a modular activation function, where the EAs are used to train weights, thresholds and choice of activation function. The inputs and output from the modular activation function consist of discrete states as opposed to real values. It is considered a modular activation function, since a neuron's behaviour could be completely altered by changing the selected activation function while holding the *modular* weights constant. The modular neuron could assume one of four different activation functions listed below:

$$\begin{aligned} \phi_1 : s_{out} &= \begin{cases} 0, & \text{if } p(x) \leq t_1 \\ 1, & \text{if } p(x) > t_1 \end{cases} & \phi_3 : s_{out} &= \begin{cases} 0, & \text{if } t_2 < p(x) < t_1 \\ 1, & \text{otherwise} \end{cases} \\ \phi_2 : s_{out} &= \begin{cases} 0, & \text{if } p(x) \geq t_2 \\ 1, & \text{if } p(x) < t_2 \end{cases} & \phi_4 : s_{out} &= \begin{cases} 0, & \text{if } p(x) > 0.5 \\ \text{rand}(0,1), & \text{if } p(x) = 0.5 \\ 1, & \text{if } p(x) < 0.5 \end{cases} \end{aligned} \quad (3)$$

These threshold functions maybe summarized in a single analytical expression as:

$$\phi = (1 - k_1)[(1 - k_2)\phi_1 + k_2\phi_2] + k_1[(1 - k_2)\phi_3 + k_2\phi_4] \quad (4)$$

Each neuron outputs one of two states $s \in S = \{0, 1\}$, and the activation function is thus encoded in the genome by k_1, k_2 and the threshold parameters $t_1, t_2 \in \square$, where $p(x)$ is defined as follows:

$$p(x) = \frac{\sum_i w_i x_i}{\sum_i x_i} \quad (5)$$

w_i is a neuron weight and x_i is an element of the input state vector. With two threshold parameters, a single neuron could be used to simulate AND, OR, NOT and XOR functions. The assumption is that a compact yet sufficiently complex (functional) neuron will speed up evolutionary training since this will reduce the need for more hidden layers and thus result in smaller networks.

3.2.2 Experiments

As mentioned above, we find that for the 3×3 tiling formation task, a lookup table architecture is found to be intractable (Fig. 8, top left). The CA lookup table architecture appears to fall victim to the bootstrap problem, since EAs are unable to find an incrementally better solution during the early phase of evolution resulting in search stagnation. In contrast, ETDN architectures can successfully solve this version of the tiling formation task and outperform other regular neural network architectures (regardless of the activation function used). Analysis of a typical solution (for ETDN with 16 expert nets) suggests decision neuron assigns expert networks not according to 'recognizable' *distal* behaviours but as *proximal* behaviours (organized according to proximity in sensor space) (Nolfi, 1997). This process of expert network assignment is evidence of task decomposition, through role assignment (Fig. 8, bottom right).

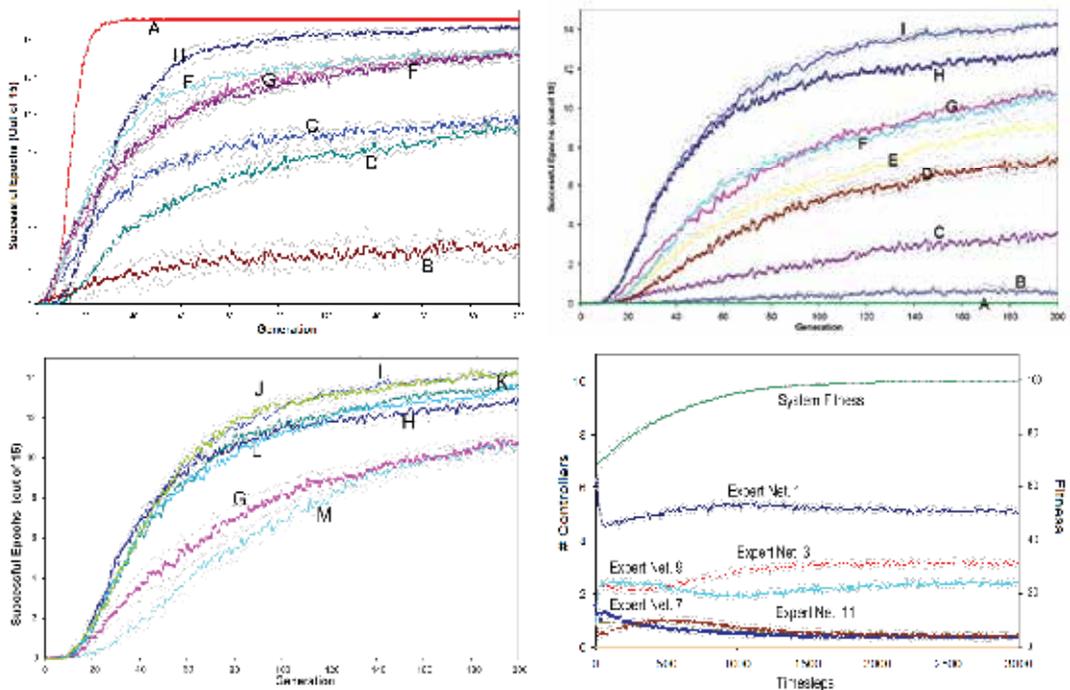


Figure 8. Evolutionary performance comparison, 2×2 (Top Left), 3×3 (Top Right, Bottom Left) tiling formation task, averaged over 120 EA runs. (Bottom Right) System activity for BRL (16 Expert Nets) (A) CA Look-up Table, (B) ESP (using Non-Emergent Net), (C) Non-Emergent Net (Sigmoid), (D) ETDN (2 Expert Nets, Sigmoid), (E) Non-Emergent Net. (Threshold), (F) ETDN (2 Expert Nets, Threshold), (G) Non-Emergent Net. (Modular), (H) ETDN (2 Expert Nets, Modular), (I) BRL (16 Expert Nets, Modular), (J) BRL (32 Expert Nets, Modular), (K) BRL (8 Expert Nets, Modular), (L) BRL (4 Expert Nets, Modular), (M) BDT (4 Expert Nets, Modular)

It should be noted that the larger BRL architecture, with more expert networks outperformed (or performed as well as) the smaller ones (evident after about 80 generations) (Fig. 8, bottom left). It is hypothesized that by increasing the number of expert networks, competition among candidate expert networks is further increased thus improving the chance of finding a desired solution. However, as the number of expert networks is increased (beyond 16), the relative improvement in performance is minimal, for this particular task.

ETDN architectures also have some limitations. For the simpler 2×2 tiling pattern formation task, a CA lookup table approach evolves desired solutions faster than the neural network architectures including ETDNs (Fig. 8, top right). This suggests that ETDNs may not be the most efficient strategy for smaller search spaces (2^{486} candidate solutions for the 2×2 tiling formation task versus 2^{4374} for the 3×3 version). Our conventional ETDN architecture consisting of a single threshold activation function evolves more slowly than the non-emergent architectures. The ETDN architectures include an additional 'overhead', since the evolutionary performance is dependent on the evolution (in tandem) of the expert networks and decision neurons resulting in slower performance for simpler tasks.

However, the ETDN architecture that combines the modular activation function outperforms all other network architectures tested. The performance of the modular neurons appears to partially offset the ‘overhead’ of the bigger ETDN architecture. A ‘richer’ activation function set is hypothesized to improve the ability of the decision neurons to switch between suitable expert networks with fewer mutational changes.

3.3 Walking Gait

For the walking gait task (Eaton et al., 2000); (Barfoot et al., 2006), a network of leg based controllers forming a hexapod robot (Fig. 9) need to find a suitable walking gait pattern that enables the robot to travel forward. We use Evolutionary Algorithms on hardware, to coevolve a network of CA walking controllers for the hexapod robot, *Kafka*. The fitness is simply the distance travelled by the robot, measured by an odometer attached to a moving treadmill. The robot has been mounted on an unmotORIZED treadmill in order to automatically measure controller performance (for walking in a straight line only). As with other experiments, the fitness measure is a global one and does not explicitly shape for a particular walking gait pattern, rather we seek the emergence of such behaviours through multiagent coordination amongst the leg controllers.

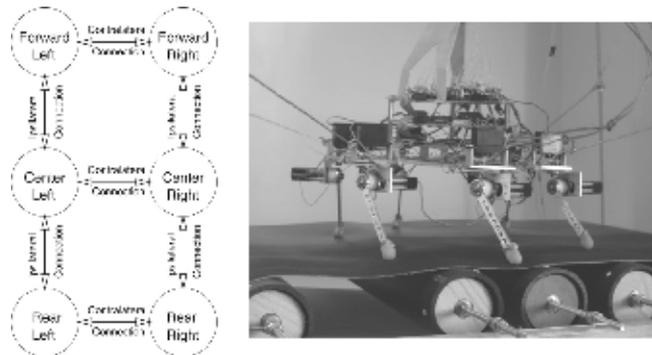


Figure 9. (Left) Behavioural coupling between legs in stick insects (Cruse, 1990). (Right) *Kafka*, a hexapod robot, and treadmill setup designed to evolve walking gaits

3.2.1 Network of Cellular-Automata Controllers

According to neurobiological evidence (Cruse, 1990), the behaviour of legs in stick insects is locally coupled as in Fig. 9 (left). This pattern of ipsilateral and contralateral connections will be adopted for the purposes of discussion although any pattern could be used in general (however, only some of them would be capable of producing viable walking gaits). The states for *Kafka*'s legs are constrained to move only in a clockwise, forward motion. The control signals to the servos are absolute positions to which the servos then move as quickly as possible. Based on the hardware setup, we make some assumptions, namely that the output of each leg controller is independent and discrete. This is in contrast to use of a central pattern generator to perform coordination amongst the leg controllers (Porcino, 1990). This may be done by way of a set of basis behaviours. Rather than specify the actuator positions (or velocities) for all times, we assume that we may select a simple behaviour from a finite predefined palette. The actual construction of the behaviours requires careful consideration but is also somewhat arbitrary.

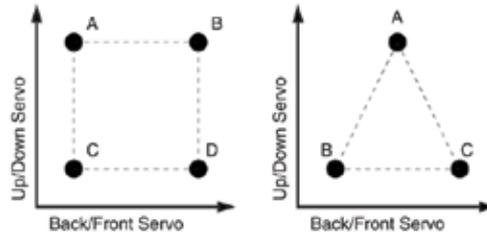


Figure 10. Example discretizations of output space for 2 degree of freedom legs into (Left) 4 zones and (Right) 3 zones

Here the basis behaviours will be modules that move the leg from its current zone (in output space) to one of a finite number of other zones. Fig. 10 shows two possible discretizations of a two-degree-of-freedom output space (corresponding to a simple leg) into 4 or 3 zones. Execution of a discrete behaviour does not guarantee the success of the corresponding leg action due to terrain variability. This is in contrast to taking readings of the leg's current zone, which gives an accurate state (local feedback signal) of the current leg position. The only feedback signal available for the discrete behaviour controller is a global one, the total distance travelled by the robot. The challenge therefore is to find an appropriate arbitration scheme which takes in a discrete input state, d , (basis behaviours of self and neighbours) and outputs the appropriate discrete output, o , (one of M basis behaviours) for each leg. One of the simpler solutions is to use separate lookup tables similar to cellular automata (CA) for each leg controller.

3.2.1 Experiments

Decentralized controllers for insect robots offer a great deal of redundancy, even if one controller fails, the robot may still limp along under the power of the remaining functional legs. The cellular automata controller approach was successfully able to control *Kafka*, a hexapod robot, and should extend to robots with more degrees of freedom (keeping in mind scaling issues). Coevolution resulted in the discovery of controllers comparable to the tripod gate (Fig. 11, 12). One advantage of using cellular automata, is that it requires very few real-time computations to be made (compared to a dynamic neural network approaches). Each leg is simply looking up its behaviour in a table and has wider applicability in hardware. The approach easily lends itself to automatic generation of controllers as was shown for the simple examples presented here.

We found that a coevolutionary technique using a network of CAs were able to produce distributed controllers that were comparable in performance to the best hand-coded solutions. In comparison, reinforcement learning techniques such as cooperative Q-learning, was much faster at this task (e.g., 1 hour instead of 8) but required a great deal more information as it was receiving feedback after shorter time-step intervals (Barfoot et al., 2006). Although both methods were using the same sensor, the reinforcement learning approach took advantage of the more detailed breakdown of rewards to increase convergence rate. The cost of this speed-up could also be seen in the need to prescribe an exploration strategy and thus determine a suitable rewarding scheme by hand. However, the coevolutionary approach requires fewer parameters to be tuned which could be advantageous for some applications.

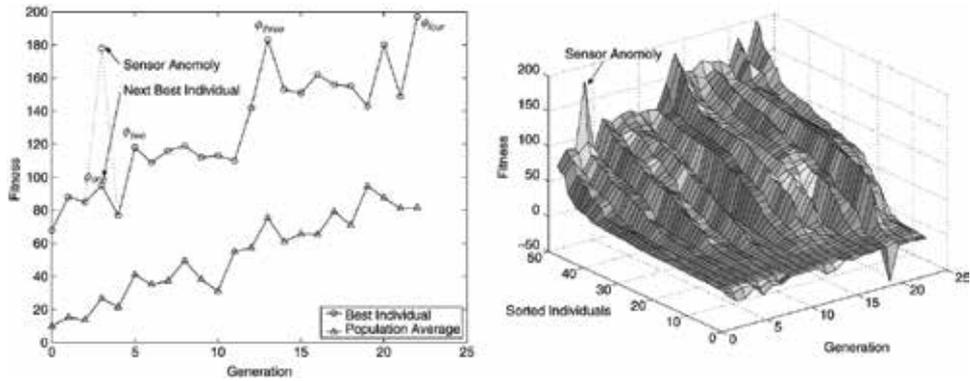


Figure 11. Convergence History of a GA Run. (Left) Best and average fitness plots over the evolution. (Right) Fitness of entire population over the evolution (individuals ordered by fitness). Note there was one data point discounted as an odometer sensor anomaly

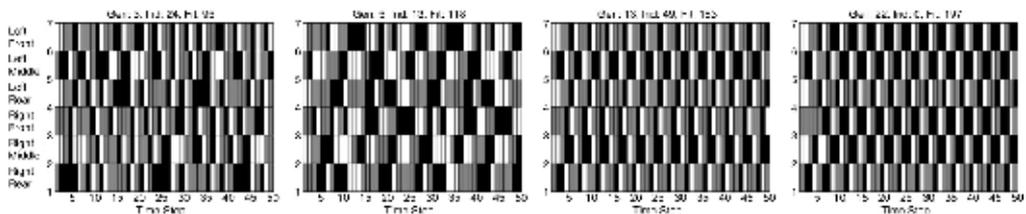


Figure 12. Gait diagrams (time histories) for the four solutions, ϕ_{one} , ϕ_{two} , ϕ_{three} , ϕ_{four} respectively. Colours correspond to each of the three leg zones in Figure 10 (right)

3.4 Resource Gathering

In this section, we look at the resource-collection task (Thangavelautham et al., 2007), which is motivated by plans to collect and process raw material on the lunar surface. Furthermore, we address the issue of *scalability*; that is, how does a controller evolved on a single robot or a small group of robots but intended for use in a larger collective of agents scale? We also investigate the associated problem of *antagonism*. For the resource gathering task, a team of robots collects resource material distributed throughout its work space and deposits it in a designated dumping area by exploiting *templates* (Fig. 13). This is in contrast to the heap formation task, where simulated robots can gather objects anywhere on the 2-D grid world. For this task, the controller must possess a number of capabilities including gathering resource material, avoiding the workspace perimeter, avoiding collisions with other robots, and forming resources into a mound at the designated location. The dumping region has perimeter markings on the floor and a light beacon mounted nearby. The two colours on the border are intended to allow the controller to determine whether the robot is inside or outside the dumping location. Though solutions can be found without the light beacon, its presence improves the efficiency of the solutions found, as it allows the robots to track the target location from a distance instead of randomly searching the workspace for the perimeter. The global fitness function for the task measures the amount of resource material accumulated in the designated location within a finite time period.

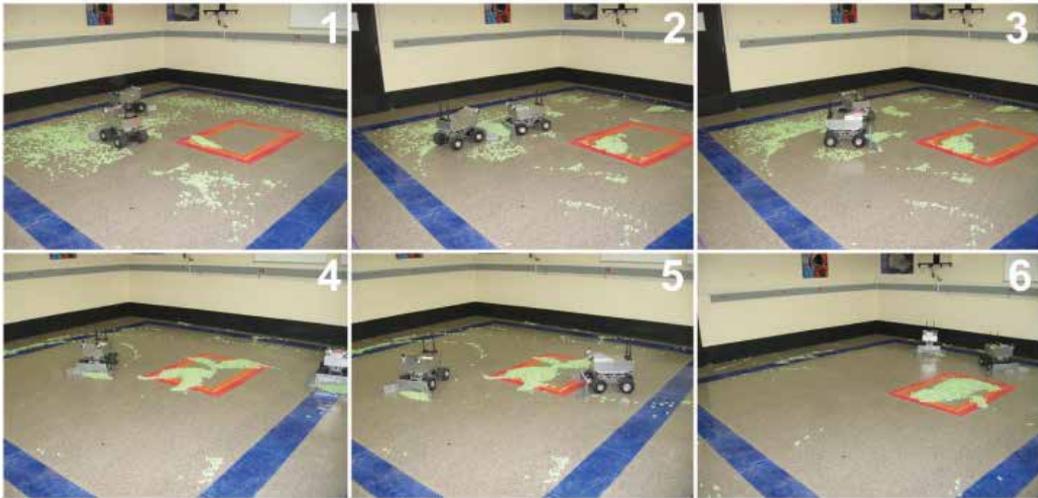
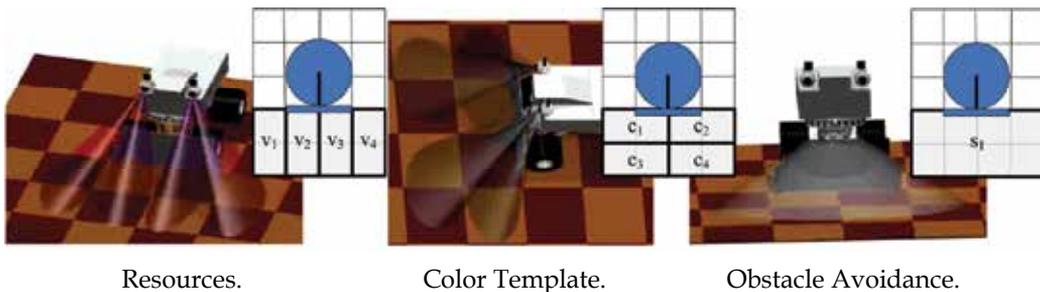


Figure 13. Snapshots of two rovers performing the resource collection task using an ANT controller. Frames 2 and 3 show the 'bucket brigade' behaviour, while frames 4 and 5 show the boundary avoidance behaviour



Resources.

Color Template.

Obstacle Avoidance.

Figure 14. Robot Input Sensor Mapping, Simulation Model Shown Inset for Resource Gathering task

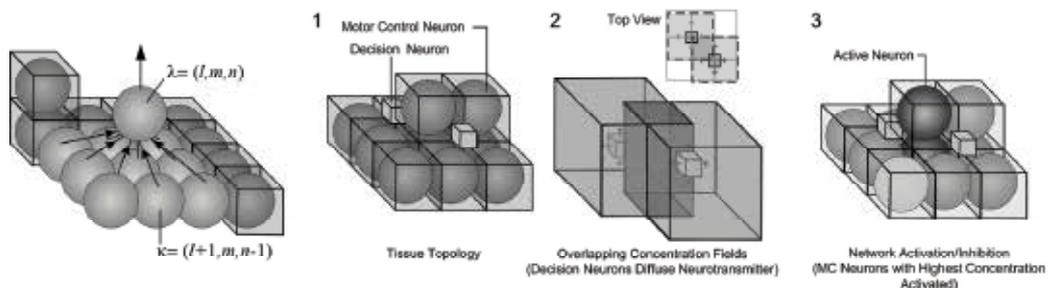
The simulated robots are modelled on a fleet of rovers designed and built at the University of Toronto Institute for Aerospace Studies. The sensor input layout for the rovers is shown in Fig. 14. For this task we consider standard fixed-topology neural networks and a variable-length topology called Artificial Neural Tissues (Thangavelautham & D'Eleuterio, 2005). ANT has been applied on a number of robotic tasks including a tiling pattern formation task, single-robot phototaxis (light homing) task and a sign following task. For the multiagent tiling pattern formation task, ANT outperformed standard fixed-topology neural networks and BRL networks (Thangavelautham & D'Eleuterio, 2005).

3.3.1 Artificial Neural Tissues

One of the limitations with a fixed topology network is that it requires *a priori* supervisor intervention in determining the network topology. An incorrect choice may result in longer training times or inability to find a desired solution. What we are after is a scalable framework that explicitly facilitates self-organized task-decomposition while requiring minimal human intervention. This would be limited to providing a generic palette of basis behaviours, sensory inputs and a global fitness function for a task at hand.

In turn, we expect an algorithm to find a suitable multirobot controller solution such as for the resource gathering task. Artificial Neural Tissues (ANT) addresses all of these stated requirements.

ANT consists of a developmental program, encoded in the genome that constructs a three-dimensional neural tissue and associated regulatory functionality. By regulatory functionality, we mean a dynamic system that can selectively activate and inhibit neuronal groups. The tissue consists of two types of neural units, decision neurons and motor-control neurons, or simply motor neurons. Both the motor neurons and decision neurons are grown by evaluating a growth program defined in the genome. The variable-length genome is modular, consisting of genes that identify characteristics of each gene. The growth program in turn reads the contents of the genes and constructs a three-dimensional lattice structure consisting of the motor control and decision neurons. Mutations to the growth program result in addition of new miscopied genes (a means of gene replication) corresponding to formation of new neurons within the tissue or inhibition of existing genes.



Synaptic Connections.

Coarse Coding.

Figure 15. (Left) Synaptic Connections between Motor Neurons from Layer $l+1$ to l . (Right) Activated Decision Neurons Diffuse Neurotransmitter Concentration Field Resulting in Activation of Motor Control Neurons with Highest Activation Concentration

Synaptic connections between the motor neurons are local, with up to 9 neighbouring motor neurons feeding from adjoining layers as shown in Fig. 15 (left). Decision neurons are fully connected to all the sensory input; however, these neurons do not share synaptic connections (wired electrical connections) with the motor neurons. Motivated by biological evidence of chemical communication in the nervous system, the decision neurons diffuse neurotransmitters chemicals, forming neurotransmitter fields as shown in Fig. 15 (right). One may think of the neurotransmitter fields as a form of wireless (as opposed to wired) communication between neurons. Motor neurons enveloped by superpositioning of multiple neurotransmitter fields and above a prescribed neurotransmitter density are activated, while the remaining motor neurons are inhibited as shown in Figure 15 (right). Through the coordinated interaction of multiple decision neurons, one observes coarse-coding of the neurotransmitter fields, selecting for wired networks of motor neurons. These selected networks of motor neurons in effect resemble the 'expert networks' from the ETDN architecture but are dynamic. These dynamic interactions result in new neurotransmitter fields being formed or dissipation of existing ones depending on sensory input being fed into the decision neurons.

3.3.2 Experiments

Fig. 16 (left) shows the fitness (population best) of the overall system evaluated at each generation of the artificial evolutionary process for the resource gathering task. The performance of a fixed-topology, fully-connected network with 12 hidden and output neurons is also shown in Fig. 16 (right). While this is not intended as a benchmark network, in a fixed-topology network there tends to be more ‘active’ synaptic connections present (since all neurons are active), and thus it takes longer for each neuron to tune these connections for all sensory inputs.

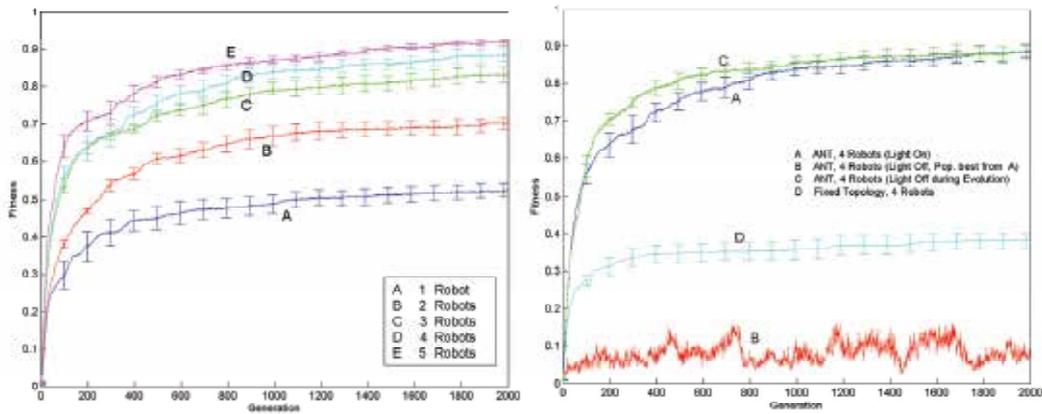


Figure 16. (Left) Evolutionary performance comparison of ANT-based solutions for 1 to 5 robots (averaged over 30 EA runs). (Right) Evolutionary performance with 4 robots for fixed topology and with light beacon off (averaged over 30 EA runs). Error bars indicate one standard deviation

The results with ANT controllers in Figure 16 (left) show that performance increases with the number of robots. With more robots, each robot has a smaller area to cover in trying to gather and dump resources. The simulation runs indicate that a point of diminishing returns is eventually reached. Beyond this point, additional robots have a minimal effect on system performance with the initial resource density and robot density kept constant. The evolutionary process enables the decomposition of a goal task based on a global fitness function, and the tuning of behaviours depending on the robot density.

The behavioural activity of the controllers (see Fig. 17) shows the formation of small networks of neurons, each of which handles an individual behaviour such as dumping resources or detecting visual templates (boundary perimeters, target area markings, etc.). Localized regions within the tissue do not exclusively handle these specific distal behaviours. Instead, the activity of the decision neurons indicate distribution of specialized ‘feature detectors’ among independent networks.

The ANT controllers discover a number of emergent behaviours including learning to pass resource material from one individual to another during an encounter, much like a ‘bucket brigade.’ This technique improves the overall efficiency of system as less time is spent travelling to and from the dumping area. In addition, ‘bucket brigades’ reduce *antagonism* since there are fewer robots to avoid at once within the dumping area.

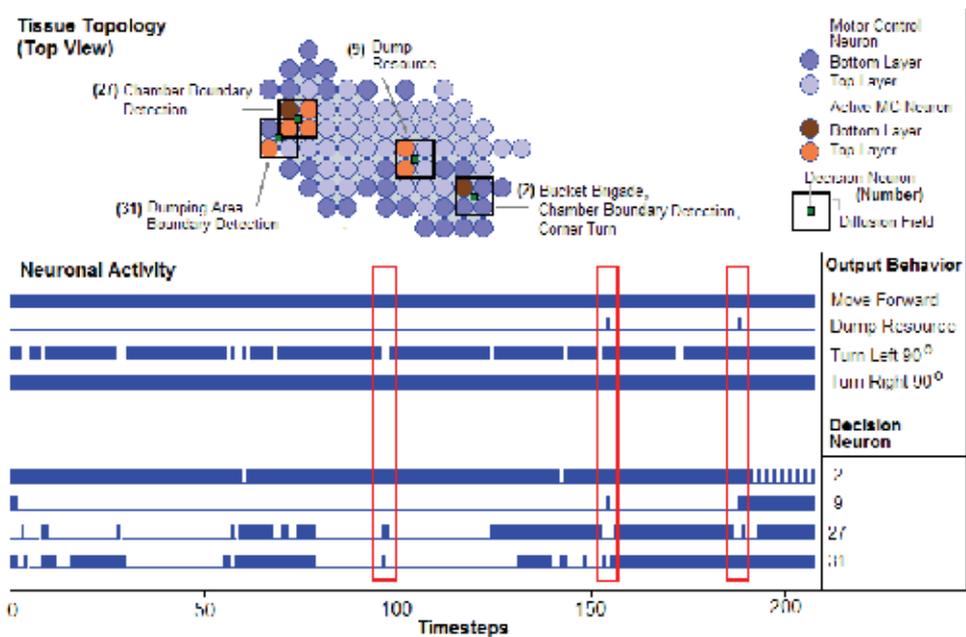


Figure 17. ANT tissue topology and neuronal activity of a select number of decision neurons. These decision neurons in turn ‘select’ (excite into operation) motor control neurons within its diffusion field

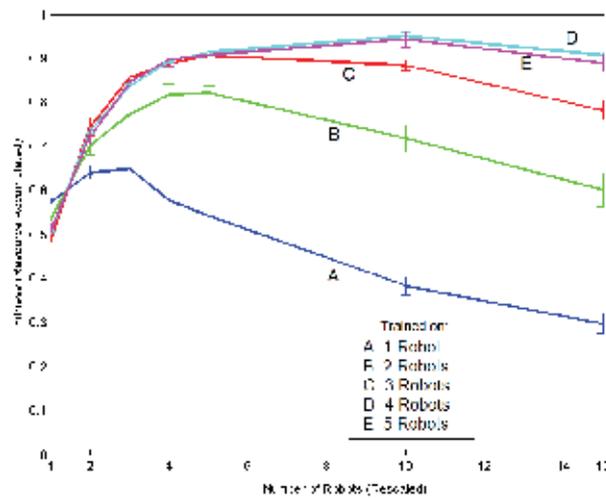


Figure 18. Scaling of ANT-based solutions (population best taken after 2000 generations) from 1 to 5 robots

3.3.3 Controller Scalability and Hardware Demonstrations

We also examine the scalability of the fittest evolved controllers by varying the density of simulated robots, while holding the resource density constant (Fig. 18). Taking the controller evolved for a single robot and running it on a multirobot system shows limited

performance improvement. In fact, using four or more robots results in a decrease in performance, due to the increased antagonism created.

The scalability of the evolved solution depends in large part on the number of robots used during the training runs. The single-robot controller expectedly lacks the cooperative behaviour necessary to function well within a multiagent setting. For example, such controllers fail to develop ‘robot collision avoidance’ or ‘bucket brigade’ behaviours and perform poorly when rescaled to multiple robots. Similarly, the robot controllers evolved with two or more robots perform demonstrably better when scaled up, showing that the solutions are dependent on cooperation among the robots. To limit the effects of antagonism, controllers need to be trained under conditions in which the probability of encounters among robots is sufficiently high.

Some of the fittest evolved controllers were also tested on a fleet of nonholonomic robots; snapshots are shown in Fig. 13. The advantage of the ANT framework with generic basis behaviours is that the solutions can be easily ported to various other hardware platforms, provided actual basis behaviours on hardware match those in simulation.

4. Discussion

The use of a global fitness function shows the possibility of training multirobot controllers with limited supervision to perform self-organized task decomposition. A global fitness function encourages solutions that improve system performance without explicitly biasing for a particular task decomposition strategy. This is advantageous for use with multirobot controllers, where it is often easier to define the global goals over required local coordination behaviours that require task specific information. In addition, such an approach facilitates discovery of novel behaviours that otherwise may be overlooked by a human designer. Such novel behaviours include use of ‘bucket brigade’ behaviours for the multirobot resource gathering task, ‘error correction’ for the tiling formation task and incremental merging of the object piles for the heap formation task. For the walking gait task, use of a global fitness function facilitated discovery of the local leg behaviours and resultant global gait coordination without the need for multistaged evolution.

While some of the behaviours from evolved controllers may display novel attributes, quantitatively interpreting such solutions remains difficult. This issue is of added importance because we are interested in how a set of local behaviours give rise to emergent global behaviours for multirobot control. As Dawkins (1986) points out, successful evolutionary solutions are not evolved per se to ease our burden of understanding such solutions. Biologically plausible techniques such as the ability to track energy consumption of the individual robots and bias for minimization of energy consumption within the fitness function also does not guarantee more decipherable solutions nor improved evolutionary training performance than without selecting for energy efficient controllers. However, such implicit techniques may well reduce any redundant behaviours that may be energetically wasteful. Alternatively, desired attributes could be explicitly encouraged through *shaping*, helping to simplify deciphering such solutions but this implies dealing with the problem of greater supervision for multirobot control and how best to perform decomposition of the task *a priori*.

Comparison of the different evolutionary techniques discussed in this chapter shows that by exploiting hierarchical modularity and regulatory functionality, controllers can overcome tractability concerns. Simple CA lookup table controllers are shown to be suitable for

solving a multiagent heap and 2×2 tiling formation tasks. Although lookup tables are monolithic, the system exploits parallelism through use of decentralized modular agents. To overcome the tractability limitations of monolithic CA lookup table controllers, we introduce modularity through decomposition of agent functionality into subsystems for the 3×3 tiling formation task or by exploiting inherent physical modularity for the walking gait task. However, such techniques often require supervisor intervention or a conducive physical setup in devising a suitable modular task decomposition scheme.

Use of neural networks is another form of modularization, where individual neurons acquire specialized functionality (via training) in solving for a given task. The neurons unlike the CA lookup tables can generalize by exploiting correlations between combinations of sensory input. Ensemble networks consist of a hierarchical modularization scheme, where networks of neurons are modularized into expert networks and the gating mechanism used to arbitrate and perform selection among the expert networks. In addition, these networks can also exploit modularity within the neuron activation function. Such techniques are shown to overcome tractability issues including the 'bootstrap problem' and reduce *spatial crosstalk* without use of a supervisor-devised task decomposition schemes.

The limitation with fixed-topology ensemble networks is that they require supervisor intervention in determining the network topology and number of expert networks. Artificial Neural Tissues (ANT) can overcome the need for supervisor intervention in determining the network topologies through use of artificial regulatory systems. Within ANT, the regulatory systems dynamically activate and inhibit neuronal groups by modelling a biologically plausible coarse-coding arbitration scheme. ANT can also overcome tractability concerns affecting other variable-length topologies that lack neural regulation. Variable-length topologies that lack neural regulatory mechanisms are grown incrementally starting from a single cell as the size of the network architecture may inadvertently make training difficult (Stanley & Miikkulainen, 2001).

The techniques used to evolve controllers for the tasks discussed also rely on predetermined sensor input layout and set of predefined basis behaviours. In practice, a sensor-input layout is often constrained due to hardware limitations or based on physical characteristics of a robot and thus relying on body-brain coevolution may not always be practical. Evolving for sensor input layouts in addition to the controller may give useful insight into sensors that are truly necessary to accomplish a task. Naturally, one would devise a fitness function to encourage use of a minimal set of sensors.

From the resource gathering experiments, we observe that evolutionary solutions can attain dependence to sensors and templates exposed during training but that are not necessary to complete a task, nor improve system performance. For the resource gathering task, use of a light beacon for homing to the 'dumping area' could intuitively increase efficiency of the robots in completing a task. It should also be noted that the light beacon is not necessary as the robots can also randomly forage for the dumping area. Although the ANT controllers can become dependent on the light beacon, this doesn't translate into improved system performance. One remedy to overcoming this dependence on optional components is through use of varied training conditions, such as with and without the light beacon. Other more elaborate failure scenarios could be introduced during training to 'condition' the controllers towards handling loss of components.

The use of a predefined palette of basis behaviours is accepted as an *ad hoc* procedure. Alternately, such basis behaviours could be obtained using machine learning techniques

such as artificial evolution in stages. Nevertheless, the intent with some of these experiments is to use a generic palette of basis behaviours that are not task-specific thus reducing the need for basis-behaviour-related design decisions by the experimenter. In practice, there exists a trade off between use of hand-coded basis behaviours (microscopic behaviours) versus use of evolutionary search techniques for multirobot coordination and control (resultant macroscopic behaviours). Where design and implementation of basis behaviours is more involved than determining the coordination behaviours, this approach may have limited practical value.

Our premise based on experience with multirobot controllers is that designing the basis behaviours requires much less effort than determining the unintuitive coordination schemes that give rise to emergent group coordination behaviours. To further emphasize this point, analyses of the evolved solutions indicate they are not organized according to readily 'recognizable' *distal* behaviours but as *proximal* behaviours (organized according to proximity in sensor space) (Nolfi, 1997). One of the more promising features of the evolved multirobot controllers solution is the scalability of these solutions to a larger 'world' size. In contrast, scalability may be limited with a hand-coded solution as other robots tend to be treated as obstacles to be avoided or use of heuristics for interaction requiring ad hoc assumptions based on limited knowledge of the task domain.

5. Conclusion

This chapter has reported on a number of investigations to control and coordinate groups of robots. Through four sets of experiments, we have made a case for carefully selecting a fitness function to encourage emergent (self-organized) task decomposition by evolutionary algorithms. This can alleviate the need to have a human designer arbitrarily break a group task into subtasks for each individual robot. We feel this approach fits well within the space engineering context where mission-level goals are commonly specified and the system designer must decompose these into manageable pieces. We have shown, in limited context of our experiments, that an evolutionary algorithm can take on this system design role and automatically break down a task for implementation by a group of robots. Limited supervision also has its advantages, including the ability to discover novel solutions that would otherwise be overlooked by a human supervisor. We also show that by exploiting hierarchical modularity and regulatory functionality, controllers can overcome tractability concerns.

Our findings are encouraging but we also realize that 'evolved controllers for multirobot systems' have many hurdles to overcome before adoption on a real space exploration mission. We must clearly show the advantages of (i) using multiple robots and (ii) programming them using an evolutionary algorithm. This will certainly be a long road, but we are enthused by the fact that the number of potential applications of multirobot systems in the space exploration world is quickly growing. For example, at the time of writing, there is renewed interest in the Moon and establishing a base near the lunar South Pole. Several robotic precursor missions are being planned to select and prepare the site and then deploy equipment including solar arrays. Once built, rovers will be needed to work alongside astronauts in searching for and gathering resources such as water ice. These are precisely the types of task by which we were originally motivated and we hope that evolutionary robotics will one day aid in the exploration of other worlds.

6. References

- Barfoot, T.D. & D'Eleuterio, G.M.T. (1999). An Evolutionary Approach to Multiagent Heap Formation, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pp. 427-435, IEEE, Washington DC.
- Barfoot, T.D. & D'Eleuterio, G.M.T. (2005). Evolving Distributed Control for an Object-Clustering Task. *Complex Systems*, Vol. 15, No. 3, 183-201
- Barfoot, T.D.; Earon, E.J.P. & D'Eleuterio, G.M.T. (2006). Experiments in Learning Distributed Control for a Hexapod Robot. *Robotics and Autonomous Systems*, Vol. 54, No. 10, pp. 864-872
- Beckers, R.; Holland, O. E. & Deneubourg, J.L. (1994). From Local actions to Global Tasks: Stigmergy and Collective Robots, *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pp. 181-189, MIT Press, Cambridge, MA
- Beer, R.D. & Gallagher, J.C. (1992). Evolving Dynamic Neural Networks for Adaptive Behavior, *Adaptive Behavior*, 1, 91-122.
- Bonabeau, E., et al. (1997). Self-organization in social insects, *Trends in Ecology and Evolution*, Vol. 12, 188-193,
- Bonabeau, E.; Dorigo, M. & Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, New York, NY, pp. 183-205.
- Bongard, J. & Pfeifer, R. (2001). Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny. *Proceedings of the Genetic and Evolutionary Computation Conference*, San Francisco, CA, pp. 829-836
- Chantemargue, F.; Dagaëff, T.; Schumacher, M. & Hirsbrunner B., (1996). Implicit Cooperation and Antagonism in Multi-Agent Systems, *Internal Report*, IIUF-PAI, University of Fribourg
- Crabbe, F. & Dyer, M. (1999). Second-order networks for wall-building agents, *Proceedings of IEEE International Joint Conference on Neural Networks*, pp. 2178- 2183, IEEE
- Cruse, H. (1990). Coordination of leg movement in walking animals, *Proceedings of the Simulation of Adaptive Behaviour: From Animals to Animats*, pp. 105-109, MIT Press
- Das, R.; Crutchfield, J.P.; Mitchell, M. & Hanson, J. (1995). Evolving globally synchronized cellular automata, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 336-343, San Fransisco, CA, Morgan Kaufmann
- Dawkins, R. (1986). *The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe without Design*, Norton & Company, New York, NY
- Dellaert, F. & Beer, R. (1994). Towards an evolvable model of development for autonomous agent synthesis. *Artificial Life IV: Proceedings of the 4th International Workshop on the Synthesis and Simulation of Living Systems*, pp. 246-257, MIT Press, Cambridge, MA
- Deneubourg, J. L., et al. (1991). The dynamics of collective sorting: Robot-like ants and ant-like robots, *Proceedings of the Simulation of Adaptive Behaviour: from Animals to Animats*, pp. 356-363, Paris, France, MIT Press, Cambridge, MA
- Deneubourg, J.L. (1977). Application de l'ordre par fluctuations `a la description de certaines 'etapes de la construction du nid chez les termites, *Insectes Sociaux*, Vol. 24, pp. 117-130
- Dorigo, M. & Colombetti, M. (1998). *Robot Shaping: An Experiment in Behavior Engineering*, MIT Press, Cambridge, MA

- Earon, E.J.P.; Barfoot, T.D. & D'Eleuterio, G.M.T. (2000). From the Sea to the Sidewalk: The Evolution of Hexapod Walking Gaits by a Genetic Algorithm, *Proceedings of the International Conference on Evolvable Systems*, pp. 51-60, Edinburgh, Scotland
- Earon, E. J. P.; Barfoot, T.D. & D'Eleuterio, G.M.T. (2001). Development of a Multiagent Robotic System with Application to Space Exploration, *Proceedings of the International Conference on Advanced Intelligent Mechatronics*, pp. 1267-1272, Como, Italy, IEEE, Washington, D.C.
- Federici, D. & Downing, K. (2006). Evolution and Development of a Multicellular Organism: Scalability, Resilience, and Neutral Complexification, *Artificial Life*, Vol. 12, pp. 381-409
- Grassé, P. (1959) La reconstruction du nid les coordinations interindividuelles; la theorie de stigmergie, *Insectes Sociaux*, Vol. 35, pp. 41-84
- Groß, R. & Dorigo, M. (2003). Evolving a Cooperative Transport Behavior for Two Simple Robots, *Proceedings of the 6th International Conference on Artificial Evolution*, pp. 305-317, Springer-Verlag, Berlin
- Gruau, F., (1995). Automatic definition of modular neural networks, *Adaptive Behaviour*, Vol. 3, No. 2, pp. 151-184.
- Jacob, R.; Jordan, M. & Barto, A. (1991). Task decomposition through competition in a modular connectionist architecture. *Cognitive Science*, Vol. 15, pp. 219-250
- Kitano, H. (1990). Designing Neural Networks using genetic Algorithm with Graph Generation System, *Complex Systems*, Vol. 4, pp. 461-476
- Kube, R. & Zhang, H. (1996). The use of perceptual cues in multirobot box-pushing. *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2085-2090, IEEE, Washington DC
- Matarić, M.J. et al. (1995). Cooperative Multi-Robot Box Pushing, *Proceedings of IEEE International Conference on Intelligent Robots and Systems*, pp. 556-561, IEEE, Washington DC
- Matarić, M.J. (1997). Behaviour-based control: Examples from navigation, learning, and group behaviour, in Software Architectures for Physical Agents, *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 9, No. 2, pp. 232-336
- Mitchell, M.; Crutchfield, J. & Das, R. (1996). Evolving cellular automata with genetic algorithms, a review of recent work. *Proceedings of the 1st International Conference on Evolutionary Computation and Its Applications*, Russian Academy of Sciences
- Nolfi, S. & Floreano, D. (2000). *Evolutionary Robotics : The Biology, Intelligence, and Technology of Self-Organizing Machines*, MIT Press, Cambridge, MA, pp. 13-15
- Nolfi, S. (1997). Using emergent modularity to develop control systems for mobile robots. *Adaptive Behavior*, Vol. 3, No. 4, pp. 343-364
- Parker, C.; Zhang, H. & Kube, R. (2003). Blind Bulldozing: Multiple Robot Nest Construction, *Proceedings of the 2003 IEEE International Conference on Intelligent Robots and Systems*, pp. 2010-2015, IEEE, Washington DC
- Parker, G. & Li, Z. (2003). Evolving Neural Networks for Hexapod Leg Controllers, *Proceedings of the 2003 IEEE International Conference on Intelligent Robots and Systems*, pp. 1376- 1381, IEEE, Washington DC
- Porcino, N. (1990). Hexapod Gait Control by Neural Network, *Proceedings of the International Joint Conference on Neural Networks*, pp. 189-194, San Diego, CA, 1990

- Roggen D. & Federici D. (2004). Multi-cellular Development: Is There Scalability and Robustness to Gain?, *Proceedings of 8th Parallel Problem Solving from Nature Conference*, pp. 391-400, Springer-Verlag, Berlin
- Sims, K. (1994). Evolving 3D Morphology and Behavior by Competition, *Artificial Life IV: Proceedings of the 4th International Workshop on the Synthesis and Simulation of Living Systems*, pp. 28-39, MIT Press, Cambridge, MA
- Stanley, K. & Miikkulainen, R. (2002). Continual Coevolution through Complexification, *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 113-120, Morgan Kaufmann, San Francisco, CA
- Stone, P. & Veloso, M. (2000). Layered Learning, *Proceedings of the 11th European Conference on Machine Learning*, pp. 369-381
- Stewart, R. & Russell, A. (2003). Emergent structures built by a minimalist autonomous robot using a swarm- inspired template mechanism, *Proceedings of the 1st Australian Conference on Artificial Life*, pp. 216-230
- Thangavelautham, J. & D'Eleuterio, G.M.T. (2004). A Neuroevolutionary Approach to Emergent Task Decomposition, *Proceedings of the 8th Parallel Problem Solving from Nature Conference*, pp. 991-1000, Springer-Verlag, Berlin
- Thangavelautham, J.; Barfoot, T.D. & D'Eleuterio G.M.T. (2003). Coevolving Communication and Cooperation for Lattice Formation Tasks, *Advances In Artificial Life: Proceedings of the 7th European Conference on Artificial Life*, pp. 857-864, Springer, Berlin
- Thangavelautham, J. & D'Eleuterio, G.M.T. (2005). A Coarse-Coding Framework for a Gene-Regulatory-Based Artificial Neural Tissue, *Advances In Artificial Life: Proceedings of the 8th European Conference on Artificial Life*, pp. 67-77, Springer, Berlin
- Thangavelautham, J.; Smith, A.; Boucher, D.; Richard, J. & D'Eleuterio, G.M.T. (2007). Evolving a Scalable Multirobot Controller Using an Artificial Neural Tissue Paradigm, *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 77-84, IEEE, Washington DC
- Thomas, L. (1974). *The Lives of a Cell: Notes of a Biology Watcher*, Penguin, USA
- Wawerla, J.; Sukhatme, G. & Mataric, M. (2002). Collective construction with multiple robots, *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2696-2701, IEEE, Washington, DC
- Wilson, M.; Melhuish, C.; Franks, A. & Scholes, S. (2004). Algorithms for building annular structures with minimalist robots inspired by brood sorting in ant colonies, *Autonomous Robots*, Vol. 17, pp. 115-136

Learning by Experience and by Imitation in Multi-Robot Systems

Dennis Barrios-Aranibar, Luiz M. G. Gonçalves and Pablo Javier Alsina
*Universidade Federal do Rio Grande do Norte
Brazil*

1. Introduction

With the increasing number of robots in industrial environments, scientists/technologists were often faced with issues on cooperation, coordination and collaboration among different robots and their self governance in the work space. This has led to the development of systems with several cooperative robotic agents. (Kim et al., 1997b). Generally, a system with several robotic agents (multi-robot system) is composed by two or more robots executing a task in a cooperative way (Arai and Ota, 1992).

Coordination, collaboration and cooperation are three terms used without distinction when working with multi-agent and multi-robot systems. In this work, we adopt a definition proposed by Noreils (Noreils, 1993) in which cooperation occurs when several agents (or robots) are gathered together so as to perform a global task. Coordination and collaboration are two forms of cooperation (Botelho and Alami, 2000). Coordination occurs when an entity coordinates its activity with another, or it synchronizes its action with respect to the other entity, by exchanging information, signals, etc. And, collaboration occurs when two or more agents decompose a global task in subtasks to be performed by each specific agent.

Generally, the solution for problems using multi-agent and multi-robot systems is divided into stages. When talking about autonomous robots, two of these stages are the task allocation stage and the task execution stage. Task allocation should be done so that all components (agents or robots) in the system are used and the problem is completely solved. The task execution stage itself should be performed so that the agents do not interfere to each other (coordination and/or collaboration) when solving the problem. Traditionally, both stages are carried out independently. In the task allocation stage, it is defined if the agents will collaborate to each other or if they will coordinate their activities. In the task execution stage, collaboration and/or coordination are effectively done.

In the literature, each stage is implemented using different techniques. The task allocation stage can be implemented using centralized or decentralized approaches (Le Pape, 1990). Centralized approaches can be implemented as an optimization problem. Decentralized approaches generally use marked based approaches like the contract-net protocol (CNP) (Ulam et al., 2007) or other approaches derived from it (Botelho and Alami, 1999).

The task execution stage can be implemented in many ways. It depends of the nature of interactions between agents (Collaboration or coordination) and if agents can modify or not their strategies (Static and dynamic strategies). For example it can be implemented using

probabilistic methods (Bruce et al., 2003), fussy logic (Wu and Lee, 2004; Chen and Liu, 2002), reinforcement learning (Salustowicz et al., 1998; Wiering et al., 1999), evolutionary algorithms (Zhang and Mackworth, 2002), neural networks (Kim et al. 1997a) and others.

A static strategy is defined during the design of the robotic system and after that, it is applied to the robots by choosing roles and actions of each robot depending of the a priori defined situation. A disadvantage of this kind of strategy is that it doesn't adapt automatically to changes in requirements and can lead to a low performance of the system if situations not envisaged by the designers occur.

On the other hand, a dynamic strategy adapts itself to the environment. This kind of strategy is generally implemented with artificial intelligence techniques. Dynamic strategies can be divided in two stages: the learning and the using stage. In the learning stage, the overall system is exposed to simulated environments, where, if necessary, opponents are programmed using static strategies. In the using stage, the system does not modify the strategy parameters. Both stages can be executed one after another during all useful life of the system.

This traditional way of implementing dynamic strategies requires a predefined set of possible situations and actions. Thus, because robots already know the kind of situations they can find in the environment and also they know what actions they can execute, we denominate these traditional approaches as "learning by experience".

We conjecture that, in real world and not in minimal applications, the robots can not completely know what kind of situations they can find and also what are all the actions that they can perform, in this case, consider an action as a set of consecutive low level signals to the actuators of the robots. In this sense, we propose to combine the imitation learning approach with learning by experience in order to construct robots that really adapt to environment changes and also evolve during their useful life. Imitation learning can help the robots to know new actions and situations where it can be applied and learning by experience can help robots to test the new actions in order to establish if they really work for the whole team.

It is important to note that the concepts, algorithms, and techniques proposed and evaluated in this work are focused in the task execution stage, specifically in dynamical strategies. Also, all the concepts are valid for multi-robot and multi-agent systems. The algorithms traditionally used for implementing learning by experience approaches are explained in section 2. Between them, we choose reinforcement learning algorithms for testing our model. In section 3 we explain our approach for implementing imitation learning and in section 4 we explain how the overall process of learning is implemented. Because there are several ways or paradigms for applying reinforcement learning to multi-robot systems, we compare them in section 5. Also, results obtained when the overall process was applied to a robot soccer problem are discussed in section 6. Finally conclusions of this work are explained in section 7.

2. Learning by Experience

The idea of using agents that can learn to solve problems became popular in the area of Artificial Intelligence, specifically in applications for machine learning techniques. When working with multi-agent or multi-robot systems, learning can be implemented in the two stages, as explained in the previous section. Thus, by developing agents that learn task allocation for the first stage and agents that learn to solve their task in the second stage, we are contributing significantly to this field.

We implement dynamic strategies, using machine learning techniques. These strategies require a predefined set of possible situations and actions. Thus, because robots already know the kind of situations they can find in the environment and also know what actions they can execute, we name those traditional approaches as learning by experience approaches. Learning by experience occurs when a robot or agent modifies the parameters used for choosing actions, from a set of them, to be executed in a set of known situations. Thus, in learning by experience, robots only modify the strategy using a set of actions prior defined by the system designer.

Almost all works found in the literature deal with robots learning by experience (Bruce et al., 2003; Wu and Lee, 2004; Chen and Liu, 2002; Wiering et al., 1999; Salustowicz et al., 1998; Zhang and Mackworth, 2002; Kim et al., 1997a).

Interactions in multi-agent and multi-robot systems can be collaborative, competitive or mixed. At this point, concepts of artificial intelligence and game theory come together to be able to explain those interactions and find ways to optimize them. In game theory, interactions between agents or robots are modeled as stochastic games. When the system is composed by a unique agent, the stochastic game is a Markov Decision Process. When the system is composed by several agents but with a unique state, it is a Repetitive Game. (Shoham et al., 2007). Stochastic games are also known as markov games and are classified in zero sum games, where agents are fully competitive, and in general sum games, where agents cooperate and/or compete to each other (Filar and Vrieze, 1997). Here, we address only those learning algorithms applied to general sum stochastic games.

2.1 Common Algorithms

Several machine learning algorithms are applied to learning in multi-agent and multi-robot cooperative systems (general sum stochastic games). In machine learning, the three most common paradigms for learning are supervised learning, unsupervised learning and the reward based learning (Panait and Luke, 2005). It is well known that these models are not easily applied for learning when using multi-agent and multi-robot systems mainly because of the complexity of these systems.

2.1.1 Supervised Learning

Supervised learning algorithms work with the assumption that it is possible to indicate the best response directly to the agent. But, when working with multi-agent and multi-robot systems, it is very difficult to indicate the best response for each agent or robot. This occurs because interactions are dynamic and there is a possibility that the teammates and/or opponents change their behavior during execution. In despite, we find some works that apply this kind of algorithm directly to multi-agent systems (Goldman and Rosenschein, 1996; Wang and Gasser, 2002; Śnieżyński and Koźlak, 2006). Some hybrid systems that fuse this kind of algorithms with other machine learning algorithms can also be found. For example, Jolly et al. (2007) combine evolutionary algorithms and neural networks for decision making in multi-robot soccer systems.

2.1.2 Unsupervised Learning

Unsupervised learning algorithms try to cluster similar data, thus, they are widely used for pattern recognition and data visualisation. However, it is difficult to imagine how they can be applied to solve task allocation and execution in multi-robot, cooperative systems. In

despite, it is possible to find a few works that use this kind of learning, as the one of Li and Parker (2007) that uses the Fuzzy C-Means (FCM) clustering algorithm for detecting faults in a team of robots.

2.1.3 Reward Based Learning

The basic idea of reward based algorithms is not to indicate what the best response is, but the expected result. Thus, the robot or agent has to discover the best strategy in order to obtain the desired results. The most representative algorithms of this type are the evolutionary algorithms and the reinforcement learning algorithms. Evolutionary algorithms search the solution space in order to find the best results. This could also be done by assessing a fitness function. On the other hand, reinforcement learning algorithms estimate a value function for states or state-action pairs. This is done for defining the best policy that take advantage of these values. In this work, we focus specifically in reinforcement learning algorithms.

2.2 Requirements for Learning by Experience

In order to get a solution when using the learn by experience model in a single agent or robot we need to define: a set of pre defined states or situations where the agent can act; a set of possible actions that the agent can perform; a way of modifying parameters of action selection while actions reveal as good or bad for certain situations.

The same set is necessary, and enough, when using multi-agent or multi-robot systems. It is important to note that, although agents need a pre defined set of states or situations, abstraction is made in a way that any real situation in the environment is mapped into any pre defined state. Generally, all actions are tested in all situations during the training process in order to verify if it is good to use the action set or not.

2.3 Paradigms for Applying Reinforcement Learning Algorithms in Multi-Robot Systems

Reinforcement learning algorithms were first introduced for learning of single agents. They can also be applied in multi-agent and multi-robot systems there existing several ways of doing it. A first one is by modeling all agents or robots as a single agent. This is known as the team learning paradigm. A second model is by applying the algorithms without any modification to each agent or robot that compose the team. This is known as independent learning. The third and last traditional paradigm is by learning joint actions. It means that each agent learns how to execute an action, thinking to combining it with actions that other agents will execute.

We introduce here a new paradigm, called influence value learning, besides the three traditional paradigms explained above. In our new model, the agents learn independently, but, at the same time, each agent is influenced by the opinions that other agents have about its actions. This new approach and the three traditional ones are better explained in the following.

2.3.1 Team Learning

The paradigm of multi-agent and multi-robot systems where agents learn as a team is based in the idea of modeling the team as a single agent. The great advantage of this paradigm is that the algorithms do not need to be modified. But, in robotics applications, it can be

difficult to implement it because we need to have a centralized learning process and sensor information processing.

An example of this paradigm using reinforcement learning is the work of Kok and Vlasis (2004) that model the problem of collaboration in multi-agent systems as a Markov Decision Process. The main problem in their work and other similar works is that the applicability becomes impossible when the number of players increases because the number of states and actions increases exponentially.

The use of genetic algorithms is not much affected by the exponential growth of the number of states and actions (Sen and Sekaran, 1996; Salustowicz et al., 1998; Iba, 1999). However, the need of centralized processing and information, what is still a problem impossible to be solved in this paradigm, is one of its disadvantages.

2.3.2 Independent Learning

The problems reported in learning as a team can be solved by implementing the learning algorithms independently in each agent. Several papers show promising results when applying this paradigm (Sen et al., 1994; Kapetanakis and Kudenko, 2002; Tumer et al., 2002). However, Claus and Boutilier (1998) explored the use of independent learners in repetitive games, empirically showing that the proposal is able to achieve only sub-optimal results. The above results are important when analyzed regarding the nature of the used algorithms. It may be noted that the reinforcement learning algorithms aim to take the agent to perform a set of actions that will provide the greatest utility (greater rewards). Below that, in problems involving several agents, it is possible that the combination of optimal individual strategies not necessarily represents an optimal team strategy. In an attempt to solve this problem, many studies have been developed. An example is the one of Kapetanakis and Kudenko (2002) which proposes a new heuristic for computing the reward values for actions based on the frequency that each action has maximum reward. They have shown empirically that their approach converges to an optimal strategy in repetitive games of two agents. Also, they test it in repetitive games with four agents, where, only one agent uses the proposed heuristic, showing that the probability of convergence to optimal strategies increases but is not guaranteed (2004). Another study (Tumer et al., 2002) explores modifications for choosing rewards. The problem of giving correct rewards in independent learning is studied. The proposed algorithm uses collective intelligence concepts for obtaining better results than by applying algorithms without any modification and learning as a team. Even achieving good results in simple problems such as repetitive games or stochastic games with few agents, another problem in this paradigm, which occurs as the number of agents increase, is that traditional algorithms are designed for problems where the environment does not change, that is, the reward is static. However, in multi-agents systems, the rewards may change over time, as the actions of other agents will influence them.

In the current work, this paradigm is analyzed in comparison with joint action learning and with our approach, the influence value learning. Q-Learning is known to be the best reinforcement learning algorithm (Sutton and Barto, 1998), so, the algorithms are going to be based on it. The Q-Learning algorithm for Independent Learning (IQ-Learning) is defined by equation 1.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(s_t, a_t)) \quad (1)$$

where $Q(s_t, a_t)$ is the value of the action a_t in the state s_t , α is the learning rate ($0 \leq \alpha \leq 1$), γ is the discount rate ($0 \leq \gamma \leq 1$), s_{t+1} is the resultant state after executing the action a_t . And, r_{t+1} is the instantaneous reward obtained by executing the action a_t .

2.3.3 Joint Action Learning

One way for solving the problem of the independent learning model is learn the best response to the actions of other agents. In this context, the joint action learning paradigm appears. Each agent should learn what the value of executing their actions in combination with the actions of others (joint action) is. By intuiting a model for other agents, it must calculate the best action for actions supposed to be executed by colleagues and/or adversaries (Kapetanakis et al., 2003; Guo et al., 2007). Claus and Boutilier explored the use of this paradigm in repetitive games (Claus and Boutilier, 1998) showing that the basic form of this paradigm does not guarantee convergence to optimal solutions. However, the authors indicate that, unlike the independent learning algorithms, this paradigm can be improved if models of other agents are improved.

Other examples include the work of Suematsu and Hayashi that guarantee convergence to optimal solutions (Suematsu and Hayashi, 2002). The work of Banerjee and Sen (Banerjee and Sen, 2007) that proposes a conditional algorithm for learning joint actions, where agents learn the conditional probability of an action be executed by an opponent be optimal. Then, agents use these probabilities for choosing their future actions. The main problem with this paradigm is the number of combinations of states and actions that grows exponentially as the number of states, actions and/or agents grows.

As said before, in the current work, algorithms will be based on Q-Learning. A modified version of the traditional Q-Learning, for joint action learning, the so called JAQ-Learning algorithm, is defined by the equation 2:

$$Q_i(s_t, a1_t, \dots, aN_t) \leftarrow Q_i(s_t, a1_t, \dots, aN_t) + \alpha(r_{t+1} + \gamma \max_{a1, \dots, aN} Q_i(s_{t+1}, a1, \dots, aN) - Q_i(s_t, a1_t, \dots, aN_t)) \quad (2)$$

where a_i is the action performed by the agent i at time t ; N is number of agents, $Q_i(s_t, a1_t, \dots, aN_t)$ is the value of the joint action $(a1_t, \dots, aN_t)$ for agent i in the state s_t . r_{t+1} is the reward obtained by agent i as it executes action a_i and as other agents execute actions $a1_t, \dots, a_{i-1_t}, a_{i+1_t}, \dots, aN_t$ respectively, α is the learning rate ($0 \leq \alpha \leq 1$), γ is the discount rate ($0 \leq \gamma \leq 1$).

An agent has to decide between its actions and not between joint actions. For this decision, it uses the expected value of its actions. The expected value includes information about the joint actions and the current beliefs about other agent that is given by (Equation 3):

$$EV(s_t, ai) \leftarrow \sum_{a_{-i} \in A_{-i}} Q(s_t, a_{-i} \cup ai) * \prod_{j \neq i} Pr_t(a_{-i} j) \quad (3)$$

where ai is an action of agent i , $EV(s_t, ai)$ is the expected value of action ai in state s_t , a_{-i} is a joint action formed only by actions of other agents, A_{-i} is the set of joint actions of other agents excluding agent i , $Q(s_t, a_{-i} \cup ai)$ is the value of a joint action formed by the union of the joint action a_{-i} of all agents excluding i with action ai of agent i in state s_t and $Pr_t(a_{-i} j)$ is the probability of agent j performs action aj that is part of joint action a_{-i} in state s_t .

2.3.4 Influence Value Learning

The learning by influence value paradigm that we propose (Barrios-Aranibar and Gonçalves, 2007a; Barrios-Aranibar and Gonçalves, 2007b) is based on the idea of influencing the behaviour of each agent according to the opinion of others. Since this proposal is developed in the context of learning through rewards, then the value of the proposed solutions will be the reward of each agent and the opinion that the other players have on the solution that the agent gave individually. This opinion should be a function of reward obtained by the agents. That is, if an agent affects the other players pushing their reward below than the previously received, they have a negative opinion for the actions done by the first agent.

From the theoretical point of view, the model proposed does not have the problems related to the paradigms of team learning and joint action learning about the number of actors, actions and states. Finally, when talking about possible changes of rewards during the learning process and that the agent must be aware that the rewards may change because of the existence of other agents, authors conjecture that this does not represent a problem for this paradigm, based on experiments conducted until now.

This paradigm is based on social interactions of people. Some theories about the social interaction can be seen in theoretical work in the area of education and psychology, such as the work of Levi Vygotsky (Oliveira and Bazzan, 2006; Jars et al., 2004).

Based on these preliminary studies on the influence of social interactions in learning, we conjecture that when people interact, they communicate each other what they think about the actions of the other, either through direct criticism or praise. This means that if person A does not like the action executed by the person B, then A will protest against B. If the person B continue to perform the same action, then A can become angry and protest angrily. We abstract this phenomenon and determined that the strength of protests may be proportional to the number of times the bad action is repeated.

Moreover, if person A likes the action executed by the person B, then A must praise B. Even if the action performed is considered as very good, then A must praise B vehemently. But if B continues to perform the same action, then A will get accustomed, and over time it will cease to praise B. The former can be abstracted by making the power of praise to be inversely proportional to the number of times the good action is repeated.

More importantly, we observe that protests and praises of others can influence the behavior of a person. Therefore, when other people protest, a person tries to avoid actions that caused these protests and when other people praise, a person tries to repeat actions more times.

Inspired in this fact, in this paradigm, agents estimate the values of their actions based on the reward obtained and a numerical value called influence value. The influence value for an agent i in a group of N agents is defined by equation 4.

$$IV_i \leftarrow \sum_{j \in (1:N), j \neq i} \beta_i(j) * OP_j(i) \quad (4)$$

Where $\beta_i(j)$ is the influence rate of agent j over agent i , $OP_j(i)$ is the opinion of agent j about the action executed by agent i .

The influence rate β determine whether or not an agent will be influenced by opinions of other agents. OP is a numerical value which is calculated on the basis of the rewards that an agent has been received. Because in reinforcement learning the value of states or state-action pairs is directly related to rewards obtained in the past, then the opinion of an agent will be calculated according to this value and reward obtained at the time of evaluation (Equation 5).

$$OP_j(i) \leftarrow \begin{cases} RV_j * Pe(s(t), a_i(t)) & \text{If } RV_j \leq 0 \\ RV_j * (1 - Pe(s(t), a_i(t))) & \text{In other case} \end{cases} \quad (5)$$

Where

$$RV_j \leftarrow r_j + \max_{a_j \in A_j} Q(s(t+1), a_j) - Q(s(t), a_j(t)) \quad (6)$$

For the case to be learning the values of state-action pairs. $Pe(s(t), a_i(t))$ is the occurrence index (times action a_i is executed by agent i in state $s(t)$ over times agent i have been in state $s(t)$). $Q(s(t), a_i(t))$ is the value of the state-action pair of the agent j at time t . And, A_j is the set of all actions agent j can execute. Thus, in the IVQ-learning algorithm based on Q-Learning, the state-action pair value for an agent i is modified using the equation 7.

$$\begin{aligned} Q(s(t), a_i(t)) &\leftarrow Q(s(t), a_i(t)) + \alpha(r(t+1) + \\ &\gamma \max_{a_i \in A_i} Q(s(t+1), a_i) - Q(s(t), a_i(t)) + IV_i) \end{aligned} \quad (7)$$

where $Q(s(t), a_i(t))$ is the value of action $a_i(t)$ executed by agent i , α is the learning rate ($0 \leq \alpha \leq 1$), γ is the discount rate ($0 \leq \gamma \leq 1$). And, $r(t+1)$ is the instantaneous reward obtained by agent i .

2.4 Robotics and Reinforcement Learning

For the implementation of reinforcement learning, it is important to define the model of the environment, the reward function, the action selection policy and the learning algorithm to be used. When applying reinforcement learning algorithms to robotics, developers have to consider that the state space and the action space are infinite.

There are several proposals to implement reinforcement learning in this kind of problem. These proposals may involve a state abstraction, function approximation and hierarchical decomposition (Morales and Sammut, 2004). In the current paper, we use state abstraction for modeling the robot soccer application where this approach was tested.

Another problem is that reinforcement learning algorithms require every action to be tested in all states. And, in robotics not all actions can be used in all states. Eventually there can exist an action that must not be used in some states. Also, if it is implemented in multi-robot systems, it is important to note that eventually some states into the model of the environment are impossible to exist in real applications. We propose a solution that can be implemented by modifying reinforcement learning algorithms in order to test actions only in certain states (for example, states pre-defined by designers or where some specialist robotic system used before)

Another problem when applying reinforcement learning to robotics is that learning algorithms require so much computational time in comparison with sampling rates needed in robotics. For solving this problem, authors propose to use off-line algorithms. The last means that algorithms will be executed during the rest time of the team of robots (e.g. After finishing to execute some tasks).

3. Learning by Imitation

Learning by imitation is considered a method to acquire complex behaviors and a way of providing seeds for future learning (Kuniyoshi and Inoue, 1993; Miyamoto and Kawato,

1998; Schaal, 1999). This kind of learning was applied in several problems like learning in humanoid robots, air hockey and marble maze games, and in robot soccer, where was implemented using a Hidden Markov Model and by teaching the robots with a Q-Learning algorithm. (Barrios-Aranibar and Alsina, 2007).

3.1 Benefits of Using Imitation with Reinforcement Learning Algorithms

Benefits of using imitation learning with any algorithm of learning by experience are directly related to the requirements exposed in section 2.2. And, in the specific case of reinforcement learning, with the problems related in section 2.4. Thus, the principal benefits of using imitation learning could be:

1. Avoid the necessity of test every action in every state. For that, agents or robots have to test action only where other agents or robots previously done.
2. Diminishes the computational time needed of convergence. Because, the search space decrease as robots only use actions in states where they previously observe that actions were used.
3. Diminishes the number of state-action pairs values to storage. Thus, it means that the number of real state-action pairs will be less than the possible ones.
4. Finally, reinforcement learning algorithms will not need to have the states and actions defined a priori.

Also, it is important to note that the exposed benefits will be obtained independent of the paradigm used for implementing reinforcement learning in a multi-robot system.

3.2 Implementing Imitation Learning

For implementing the imitation learning, we propose to do it by observing other teams playing soccer for recognizing complex behaviors (Barrios-Aranibar and Alsina, 2007). First, all robots have to recognize what actions robots they are trying to imitate are performing. After that those actions have to be grouped into behaviors. And, finally by defining in which states observed robots used them, the recognized behaviors have to be included into the data base of learning robots. This approach is explained below.

3.2.1 Recognizing Actions of Other Robots

To understand this approach of learning by imitation, concepts like role, situation, action and behavior must be defined. A role is defined as the function a robot implements during it's useful life in a problem or in part of it. Each role is composed of a set of behaviors. Also, a behavior is a sequence of consecutive actions. And, an action is a basic interaction between a robot (actor) and an element in the environment, including other robots in the team. Finally, a situation is an observer description of a basic interaction between a robot and an element in the environment, including other robots in the team. This means that a situation is the abstraction that an observer makes about something other robot does.

For recognizing action of other robots, the learners have to really analyze and recognize situations. We propose to do this analysis over games previously saved. Because the application used for testing the overall approach is the robot soccer, in current work, the saved game must include positions and orientations of the robots of both teams, the ball position and scores at every sampling time.

The analysis of saved games is made with a fuzzy inference machine. Here, situations involving each robot in the analyzed team must be made. The fuzzy inference machine includes five fuzzy variables: Distance between two objects, orientation of an object with

respect to the orientation of another object, orientation of an object with respect to another object, playing time and velocity of an object with respect to the velocity of another object. For defining these fuzzy variables, values of time (sampling times), position, orientation, distance, velocity and direction were used. Figure 1 shows possible fuzzy values for each variable.

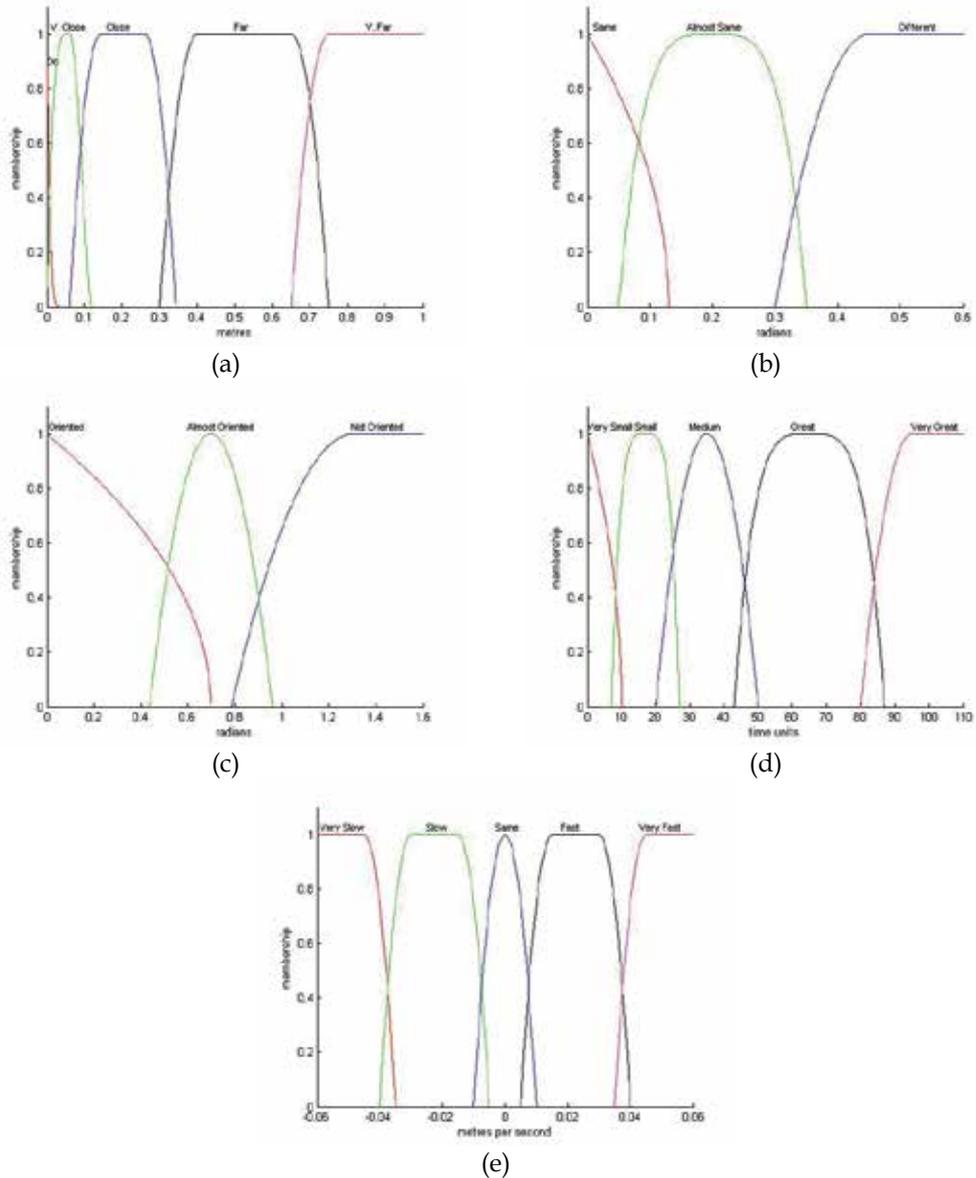


Figure 1. Fuzzy variables for recognizing situations: (a) Distance Between two Objects. (b) Orientation of an Object with Respect to Orientation of Another Object. (c) Orientation of an Object with Respect to Another Object. (d) Playing Time. (e) Velocity of an Object with Respect to the Velocity of Another Object

Distance between two objects fuzzy variable is based in the Euclidian distance between the objects positions. Orientation of an object with respect to orientation of another object fuzzy variable is based on the absolute value of the difference between the orientation angles of two objects. Orientation of an object with respect to another object fuzzy variable is based on the angular distance between two objects. Which is defined as the difference between the current orientation of the first object and the orientation needed to point to the second object. Playing time fuzzy variable is based on the sampled times of a game or a part of it. And, velocity of an object with respect to the velocity of another object fuzzy variable is based on the value of the difference between the velocity of two objects.

Code	Situation Name	Priority
001	Robot with the ball	1
011	Robot guides the ball	2
021	Robot kicks the ball	3
022	Robot loses the ball	4
023	Robot yields the ball	4
033	Robot leaves the ball	5
034	Robot moves away from the ball	10
044	Robot reaches the ball	6
045	Robot receives the ball	7
046	Robot approaches the ball	9
047	Ball hits the robot	8
057	Robot orients to the ball	11
067	Robot goes ball's X direction	12
068	Robot goes ball's Y direction	13
078	Robot orients to it's own goal	14
088	Robot approaches it's own goal	15
098	Robot moves away from it's own goal	16
108	Robot orients to adversary's goal	17
118	Robot approaches the adversary's goal	18
128	Robot moves away from adversary's goal	19
138	Robot approaches goalkeeper adversary	21
139	Robot approaches midfield adversary	21
140	Robot approaches striker adversary	21
150	Robot moves away from goalkeeper adversary	24
151	Robot moves away from midfield adversary	24
152	Robot moves away from striker adversary	24
162	Robot approaches role +1 teammate	23
163	Robot approaches role +2 teammate	23
173	Robot moves away from role+1 teammate	22
174	Robot moves away from role+2 teammate	22
184	Robot does not move	20
194	Robot moves randomly	

Table 1. Situations Defined in the Fuzzy Inference Machine

Thirty situations with fuzzy rules were defined. An additional situation called "Robot moving randomly" was defined to be used when there is a time interval that does not fulfill

restrictions of any situation. The situations are listed in table 1. Three roles were defined for the robots: Goalkeeper, midfield and striker, all of them depend on the robot position at each sampling time. Almost all situations are independent of the robot's role. The situations with codes 162, 163, 173 and 174 depend on the robot role when a situation starts. For that purpose, roles are arranged in a circular list (e.g. Goalkeeper, midfield, striker).

The recognition of a situation passes through two stages: Verification of possibility and verification of occurrence. The verification of possibility is made with a fuzzy rule called "Initial Condition". If at any time the game this rule is satisfied then the situation is marked as possible to happen. This means that in the future the rule for verification of occurrence should be evaluated ("Final Condition"). If the situation is scheduled as possible to happen, in every sampling time, in the future, the fuzzy rule "Final Condition" will be checked. The system has certain amount of time to verify this condition. If past the time limit this condition is not fulfilled, then the mark of possible to happen is deleted for that situation.

As shown in table 1, each situation has a priority for recognition on the fuzzy inference machine. Situations with priority 1 has the highest priority and situations with priority 24 have less priority.

3.2.2 Recognizing Behaviors of Other Robots

After recognizing all the situations for each of the robots of the analyzed time, the codes of these situations will be passed by self-organized maps (SOM), proposed by Kohonem. Both, the recognition of situations and the recognition of patterns of behaviours are done offline and after each training game.

At the beginning of the process, four groups of successive events are generated. They are considered all possible combinations of successive grouping of situations without changing their order, which means that each situation may be part of up to 4 groups (e.g. where it can be the first, second, third or fourth member of the group).

The groups formed are used to train a SOM neural network. After finishing the process of training, the neurons that were activated by at least 10% of the number of recognized situations are selected. Then, the groups who have activated the previously selected neurons are selected to form part of the knowledge base. To be part of knowledge base, each group or behavior pattern must have a value greater than a threshold.

The value of each group is calculated based on the final result obtained by the analyzed team. Final results could be: a goal of the analyzed team, goal of the opponent team, or end of the game. All situations recognized before a goal of the analyzed team receive a positive value α where $0 < \alpha < 1$ and t is the number of discrete times between the start of the situation and the final result. Each situation recognized before a goal of the opponent team, receives a negative value $-\alpha$. Finally, each situation recognized before the end of the game get the value of zero. The value of each group of situations is calculated using the arithmetic mean of the values of the situations.

After recognize the patterns of behaviours formed by four situations, groups of three situations are formed. The groups are formed with those situations that were not considered before (those that do not form part of any of the new behaviors entered in the knowledge base). It is important to form groups only with consecutive situations. It can not be formed groups of situations separated by some other situation. The process conducted for groups of four situations is then repeated, but this time will be considered the neurons that were activated by at least 8% of the number of recognized situations. After that, The process is

repeated again for groups of two and considering the neurons activated for at least the 6% of the number of recognized situations. Finally, those situations that were not considered in the three previous cases, form individually a behavior pattern. Again, the process is repeated considering the neurons activated for at least the 4% of the number of recognized situations. Since, to improve the learned by imitation strategy, each new behavior inserted in the knowledge base has to be tested by the learner. Each of these behaviors receive an optimistic value (e.g. The greatest value of behaviors that can be used in the state).

4. The Overall Learning Process

The process of learning becomes first by a stage of imitation for after try and see if the learned actions are valid for the robot is learning. Therefore, it is important to define the structure of the state and also what are the instant rewards in the application chosen (robot soccer). As said before, the number of states in robotic problems are infinite. To implement reinforcement learning authors opted up by state abstraction. The purpose of this abstraction is to get a set of finite states.

The state of a robot soccer game is constituted by the positions, orientations and velocities of the robots; the position, direction and velocity of the ball as well as scores of the game. The positions of the robots and the ball were abstracted in discrete variables of distance and position with reference to certain elements of the game. Orientations of the robots were abstracted in discrete variables of direction. Direction of the ball was abstracted in a discrete direction variable. The same was done with the velocity and scores. Finally, to recognize the terminal states of the game, there was set a final element in the state that is the situation of the game. Table 2 shows the configuration of a state.

Element	Quantity
Distance of robot to ball	6 (6 robots x 1 ball)
Distance of ball to goal	2 (1 ball x 2 goals)
Orientation of robot to ball	6 (6 robots x 1 ball)
Orientation of robot to goal	12 (6 robots x 2 goals)
Robot/ball position in relation to own goal	6 (6 robots)
Ball direction	1 (1 ball)
Ball velocity	1 (1 ball)
Game score	1 (1 game)
Game situation	1 (1 game)
TOTAL	36

Table 2. Abstraction of a state in a robot soccer game

To abstract the position of the robots, we consider that the important in the position of a robot is whether he is too close, near or far the ball and whether the ball is close, very close to or far from the goals. It is also important to know if the robot is before or after the ball. With these three discrete variables you can identify the position of the robot and its location within the soccer field in relation to all elements (the ball, goals and robots)

Besides recognize the location of the robot in relation to the ball, the goal and the other robots, it is important to know where he is oriented, so we can know whether he is ready to shoot the ball or go towards the own goal. Then, the orientation of the robot was abstracted in two discrete variables called orientation to the ball and orientation to the goal.

In the case of the direction of the ball, their speed and scores, there was defined discrete variables for each considering the characteristics of the soccer game. In the case of the ball direction, 8 discrete levels were defined (each level grouping 45°). Also, there were defined 3 speed levels. In the case of the scores, important is whether the team is winning, losing or drawing. Finally the situation of the game may be normal (not terminal state), fault or goal (terminal states).

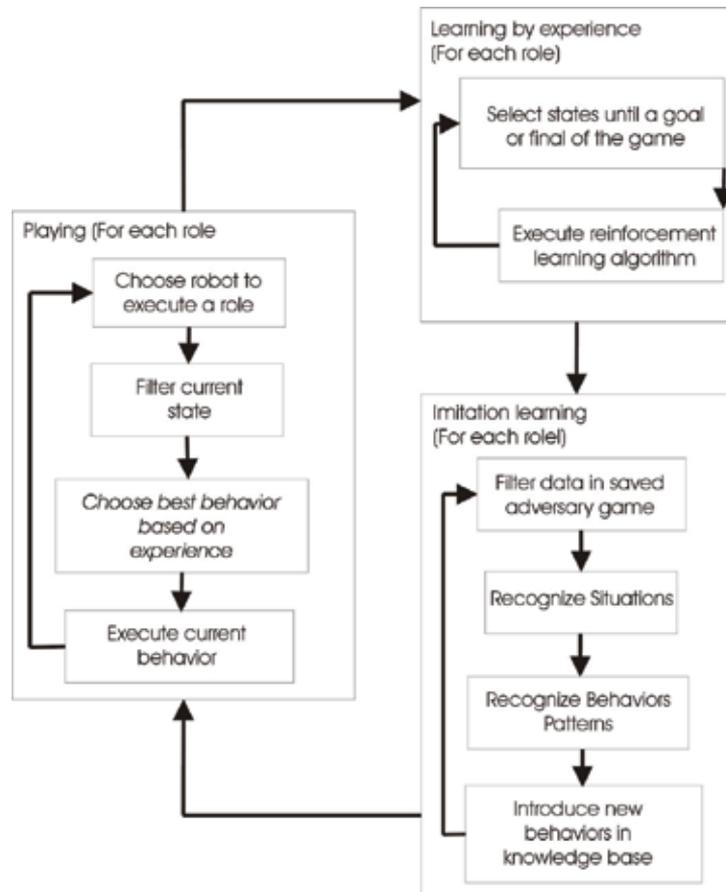


Figure 2. Training process of a team of robots learning by experience and by imitation

Considering all the possibilities of each element of the state we calculate that a robot soccer game would have 80621568 states. If we consider that the states where the situation of the game is fault or goal are only to aid in the implementation of algorithms, then we would have that the number of states decreases to 26873856. The previous value is the value of the theoretical maximum number of states in a robot soccer game. This theoretical value will be hardly achieved since not all combinations of the values of the components of the state will exist. An example of how the number of states will be reduced in practice is the case of states of the goalkeeper role. The goalkeeper role is attributed to the robot that is closest to its own goal. For states where the goalkeeper is after ball, it will be almost impossible for the other robots to be before it. Then the number of states is virtually reduced by two thirds.

Thus, considering only this case we see that the number of states for the goalkeeper reduced to 8957952.

In the case of actions in reinforcement learning, each behavior learned during imitation is considered an action in the model of reinforcement learning. Since learning by imitation is the seed for reinforcement learning, we have that reinforcement learning acts only on states and behaviors learned by the robot in previous games.

Figure 2 shows the simplified procedure of the overall training process of the control and coordination system for a robot soccer team.

Since learning of new formations, behaviors and actions, as well as the best choice for them at a particular moment, is defined by the quality of opponents teams, it is proposed that the training process is done through the interaction with human controlled teams of robots. The main advantages of this training are given by the ability of humans to learn and also by the large number of solutions that a team composed by humans could give to a particular situation.

5. What is the best Paradigm of Reinforcement Learning for Multi-Robot systems

Before assessing the paradigms of reinforcement learning for multi-robot or multi-agent systems, it is important to know that when talking about cooperative agents or robots, it is necessary that agents cooperate on equality and that all agents receive equitable rewards for solving the task. Is in this context that a different concept from the games theory appears in multi-agent systems. This is the concept of the Nash equilibrium.

Let be a multi-agent system formed by N agents. σ_i^* is defined as the strategy chosen by the agent i , σ_i as any strategy of the agent i , and Σ_i as the set of all possible strategies of i . It is said that the strategies $\sigma_1^*, \dots, \sigma_N^*$ constitute a Nash equilibrium, if inequality 8 is true for all $\sigma_i \in \Sigma_i$ and for all agents i .

$$r_i(\sigma_1^*, \dots, \sigma_{i-1}^*, \sigma_i, \sigma_{i+1}^*, \dots, \sigma_N^*) \leq r_i(\sigma_1^*, \dots, \sigma_{i-1}^*, \sigma_i^*, \sigma_{i+1}^*, \dots, \sigma_N^*) \quad (8)$$

Where r_i is the reward obtained by agent i .

The idea of Nash equilibrium, is that the strategy of each agent is the best response to the strategies of their colleagues and/or opponents (Kononen, 2004). Then, it is expected that learning algorithms can converge to a Nash equilibrium, and it is desired that can converge to the optimal Nash equilibrium, that is the one where the reward for all agents is the best.

We test and compare all paradigms using two repetitive games (The penalty problem and the climbing problem) and one stochastic game for two agents. The penalty problem, in which IQ-Learning, JAQ-Learning and IVQ-Learning can converge to the optimal equilibrium over certain conditions, is used for testing capability of those algorithms to converge to optimal equilibrium. And, the climbing problem, in which IQ-Learning, JAQ-Learning can not converge to optimal equilibrium was used to test if IVQ-Learning can do it. Also, a game called the grid world game was created for testing coordination between two agents. Here, both agents have to coordinate their actions in order to obtain positive rewards. Lack of coordination causes penalties. Figure 3 shows the three games used here.

In penalty game, $k < 0$ is a penalty. In this game, there exist three Nash equilibriums ((a0,b0), (a1,b1) and (a2,b2)), but only two of them are optimal Nash equilibriums ((a0, b0)

and (a3, b3)). When $k = 0$ (no penalty for any action in the game), the three algorithms (IQ-Learning, JAQ-Learning and IVQ-Learning) converge to the optimal equilibrium with probability one. However, as k decrease, this probability also decrease.

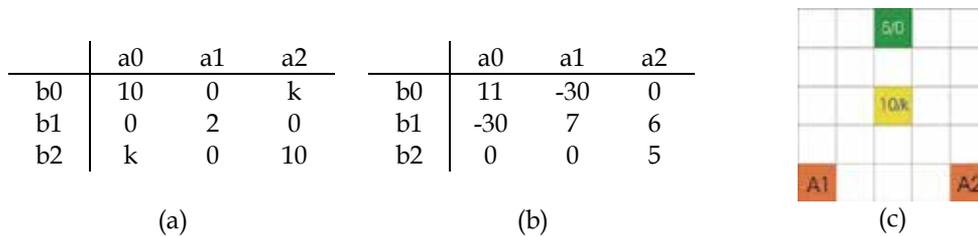


Figure 3. Games used for testing performance of paradigms for applying reinforcement learning in multi-agent systems: (a) Penalty game, (b) Climbing game, (c) Grid world game

Figure 4 compiles results obtained by these three algorithms, all of them was executed with the same conditions: A Boltzman action selection strategy with initial temperature $T = 16$, $\lambda = 0.1$ and in the case of IVQ-Learning $\beta = 0.05$. Also, a varying decaying rate for T was defined and each algorithm was executed 100 times for each decaying rate.

In this problem JAQ-Learning has the best perform. But, it is important to note also that for values of k near to zero, IVQ-Learning and IQ-Learning performs better than the JAQ-Learning, and for those values the IVQ-Learning algorithm has the best probability to converge to the optimal equilibrium.

The climbing game problem is specially difficult for reinforcement learning algorithms because action a2 has the maximum total reward for agent A and action b1 has the maximum total reward for agent B. Independent learning approaches and joint action learning was showed to converge in the best case only to the (a1, b1) action pair (Claus and Boutilier, 1998). Again, each algorithm was executed 100 times in the same conditions: A Boltzman action selection strategy with initial temperature $T = 16$, $\lambda = 0.1$ and in the case of IVQ-Learning $\beta = 0.1$ and a varying temperature decaying rate.

In relation to the IQ-Learning and the JAQ-Learning, obtained results confirm that these two algorithms can not converge to optimal equilibrium. IVQ-Learning is the unique algorithm that has a probability different to zero for converging to the optimal Nash equilibrium, but this probability depends on the temperature decaying rate of the Boltzman action selection strategy (figure 5). In experiments, the best temperature decaying rate founded was 0.9997 on which probability to convergence to optimal equilibrium (a0, b0) is near to 0.7.

The grid world game starts with the agent one (A1) in position (5; 1) and agent two (A2) in position (5; 5). The idea is to reach positions (1; 3) and (3; 3) at the same time in order to finish the game. If they reach these final positions at the same time, they obtain a positive reward (5 and 10 points respectively). However, if only one of them reaches the position (3; 3) they are punished with a penalty value k . In the other hand, if only one of them reaches position (1; 3) they are not punished.

This game has several Nash equilibrium solutions, the policies that lead agents to obtain 5 points and 10 points, however, optimal Nash equilibrium solutions are those that lead agents to obtain 10 points in four steps.

The first tested algorithm (Independent Learning A) considers that the state for each agent is the position of the agent, thus, the state space does not consider the position of the other agent. The second version of this algorithm (Independent Learning B) considers that the

state space is the position of both agents. The third one is the JAQ-Learning algorithm and the last one is the IVQ-Learning.

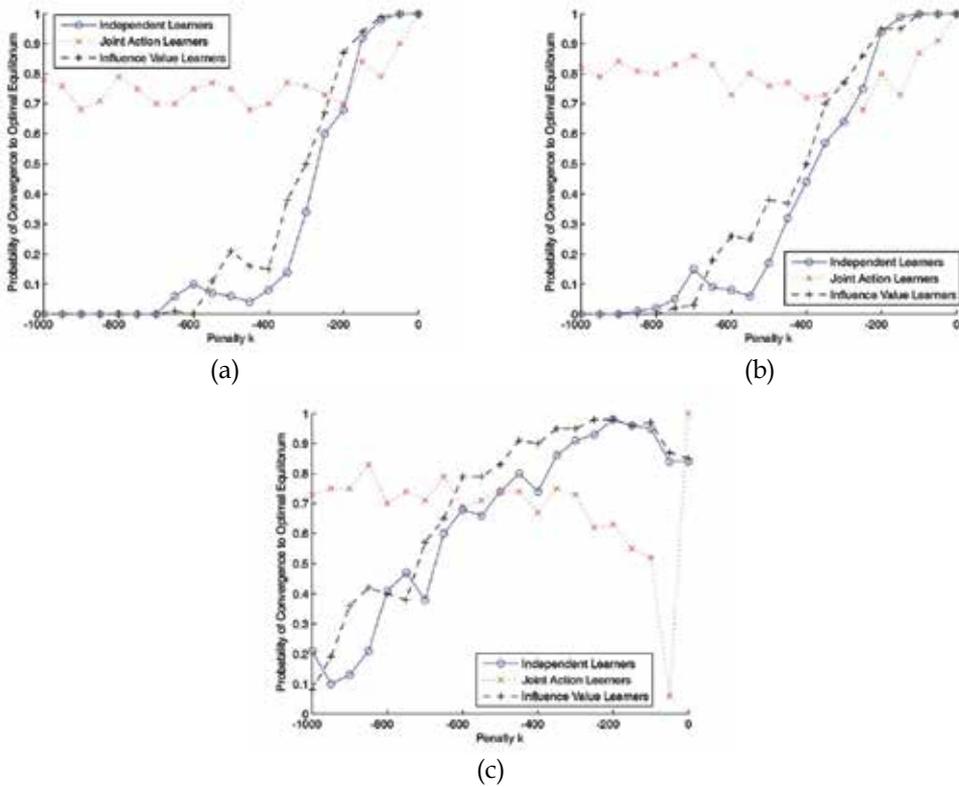


Figure 4. Probability of convergence to optimal equilibrium in the penalty game for $\lambda = 0.1$, $\beta = 0.05$ and (a) $T = 0.998 \cdot 16$, (b) $T = 0.999 \cdot 16$, and (c) $T = 0.9999 \cdot 16$

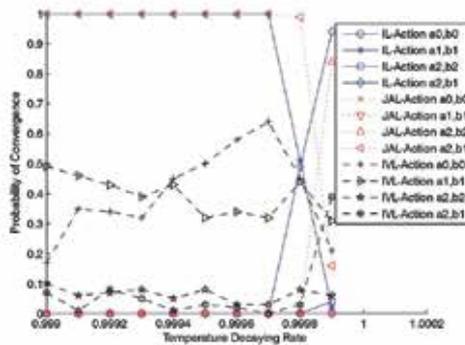


Figure 5. Probability of Convergence in Climbing Game with $\lambda = 0.1$, $\beta = 0.1$ and Variable Temperature Decaying Rate

In the tests, each learning algorithm was executed three times for each value of penalty k ($0 \leq k \leq 15$) and using five different decreasing rates of temperature T for the softmax policy

(0:99t; 0:995t; 0:999t; 0:9995t; 0:9999t). Each resulting policy (960 policies, 3 for each algorithm with penalty k and a certain decreasing rate of T) was tested 1000 of times.

Figure 6 shows the probability of reaching the position (3; 3) with $\alpha=1$, $\lambda=0.1$, $\beta=0:1$ and $T = 0:99t$. In this figure, was observed that in this problem the joint action learning algorithm has the smaller probability of convergence to the (3; 3) position. This behavior is repeated for the other temperature decreasing rates. From the experiments, we note that the Independent Learning B and our approach have had almost the same behavior. But, when the exploration rate increases, the probability of convergence to the optimal equilibrium decreases for the Independent Learners and increase for our paradigm.

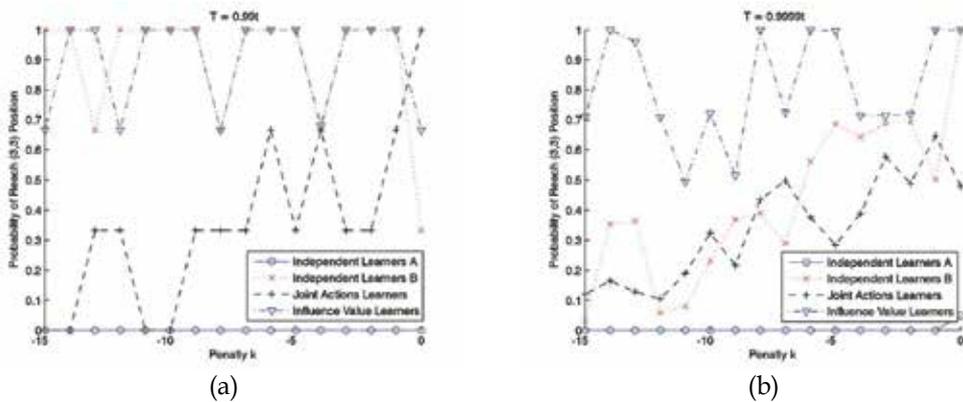


Figure 6. Probability of reaching (3,3) position for (a) $T = 0.99t$ and (b) $T = 0.9999t$

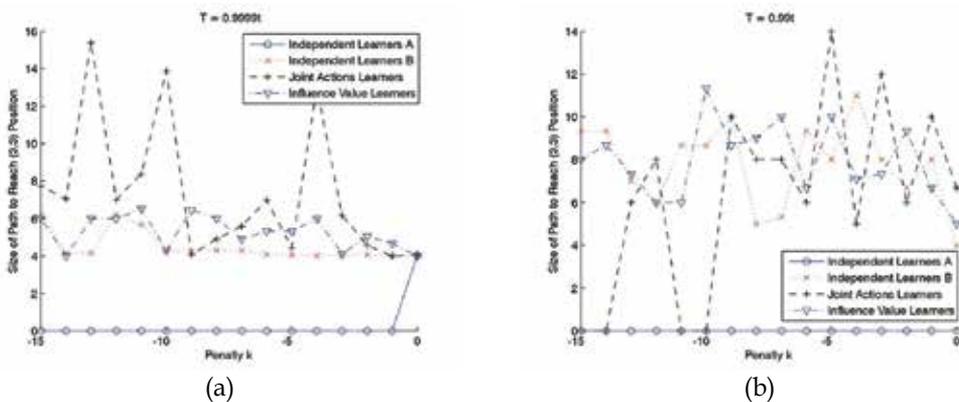


Figure 7. Size of path for reaching (3,3) position for (a) $T = 0.99t$ and (b) $T = 0.9999t$

As shown in figure 7, as more exploratory the action selection policy is, smaller is the size of the path for reaching (3; 3) position. Then, it can be concluded that when exploration increases, the probability of the algorithms to reach the optimal equilibrium increases too. It is important to note that our paradigm has the best probability of convergence to the optimal equilibrium. It can be concluded by joining the probability of convergence to the position (3; 3) and the mean size of the path for reaching this position.

6. How Performs the Proposed Learning Process

For testing the overall proposed approach, a basic reinforcement learning mechanism was chosen. This mechanism (eligibility traces) is considered as a bridge between Monte Carlo and temporal differences algorithms. Then, because the best algorithm that uses eligibility traces is the Sarsa(λ), it was used in the current work. Also, because the simplest paradigm for applying reinforcement learning to multi-robot systems is the independent learning, then, it was used for testing the overall approach. Finally, we conjecture that results obtained here validate this approach and also by using better techniques, like influence value reinforcement learning, results could be also improved.

A robot soccer simulator constructed by Adelardo Medeiros was used for training and testing our approach. Also, the system was trained in successive games of approximately eight minutes against a team of robots controlled by joystick by humans, and against a team using the static strategy developed by Yamamoto (Yamamoto, 2005).

An analysis of the amount of new actions executed in each game was conducted. Both training process (games against humans and games against the strategy of Yamamoto) were analyzed. Table 3 shows results of this analysis.

As can be seen in this table, the team that played against humans had a positive and continue evolution in the number of new actions executed during the first ten games, after these games the progress was slower, but with a tendency to increase the amount of not random actions. But the team that played against the static strategy developed by Yamamoto failed to evolve in the first ten games. This team only began to evolve from the game number eleven, where the initial state of the game was changed to a one with very positive scores (e.g. The apprentice team started winning).

Results shown in this table represent an advantage of the training against a team controlled by humans over the training against a team controlled with a static strategy. Moreover, it is important to note that both teams have a low rate of use of new actions during the game, this is due to the fact that initially, all the new states have the action "moving randomly". Also, exists a possibility that the apprentice is also learning random actions. Finally, it is important to note that this learning approach is very slow due to robots will have to test many times each action, including the random action. In this approach the knowledge base of each of the three roles starts empty and robots start using the action "moves randomly", then it is important to know how many new states and actions robots will learn after each game.

Figure 8 shows the number of states for each one of the three roles defined in this game. It compares both learning processes (against humans and against Yamamoto's strategy). Also, figure 9 shows the number of actions of each role. It is important to note that number of states increases during playing and during training. And, number of actions increases only during training.

As could be observed in these figures, the number of states and actions of the team trained against Yamamoto's strategy is greater than the other one. Also, it is important to note that despite this condition, the number of new actions executed by the team trained against humans is greater and increase over time.

In a soccer game, goals could be effect of: direct action of a player, indirect action, or error of the adversary team. An analysis of effectively made goals was conducted for knowing if learner teams really learn how to make goals (the main goal of playing soccer). For this

propose, a program developed for commenting games was used (Barrios-Aranibar and Alsina, 2007).

Game	Against Humans				Against Yamamoto's Strategy			
	Total	Random	Other	%	Total	Random	Other	%
1	3017	3017	0	0.00	3748	3748	0	0.00
2	4151	4115	36	0.87	3667	3667	0	0.00
3	4212	4168	44	1.04	3132	3132	0	0.00
4	2972	2948	24	0.81	3480	3480	0	0.00
5	2997	2973	24	0.80	3465	3465	0	0.00
6	2728	2657	71	2.60	3529	3529	0	0.00
7	3234	3162	72	2.23	4145	4145	0	0.00
8	2662	2570	92	3.46	3430	3430	0	0.00
9	3058	2886	172	5.62	3969	3969	0	0.00
10	2576	2427	149	5.78	5230	5239	0	0.00
11	3035	2812	223	7.35	4295	4198	97	2.26
12	3448	3447	1	0.03	4212	4149	63	1.50
13	3619	3464	155	4.28	2953	2842	111	3.76
14	3687	3587	100	2.71	4021	3874	147	3.66
15	3157	3071	86	2.72	4025	3942	83	2.06
16	4427	4293	134	3.03	4351	4168	183	4.21
17	3835	3701	134	3.49	4468	4273	195	4.36
18	3615	3453	162	4.48	3741	3598	143	3.82
19	4624	4497	127	2.75	4379	4173	206	4.70
20	4441	4441	0	0.00	3765	3548	217	5.76
21	4587	4422	165	3.60	4171	4047	124	2.97
22	4115	4115	0	0.00	4484	4329	155	3.46
23	4369	4369	0	0.00	4289	4178	111	2.59
24	3920	3920	0	0.00	3819	3646	173	4.53
25	3601	3447	154	4.28	4074	4074	0	0.00
26	4269	4269	0	0.00	4125	4125	0	0.00
27	4517	4347	170	3.76	3967	3967	0	0.00
28	6445	6195	250	3.88	3899	3745	154	3.95
29	3437	3346	91	2.65	4280	4081	199	4.65
30	3819	3686	133	3.48	3756	3704	52	1.38
31	4779	4779	0	0.00	3446	3294	152	4.41
32	4710	4546	164	3.48	4557	4370	187	4.10
33	3439	3285	154	4.48	4413	4203	210	4.76
34	4085	3928	157	3.84	3909	3742	167	4.27
35	3537	3454	83	2.35	3953	3776	177	4.48

Table 3. Analysis of new actions executed by learner teams using the proposed approach

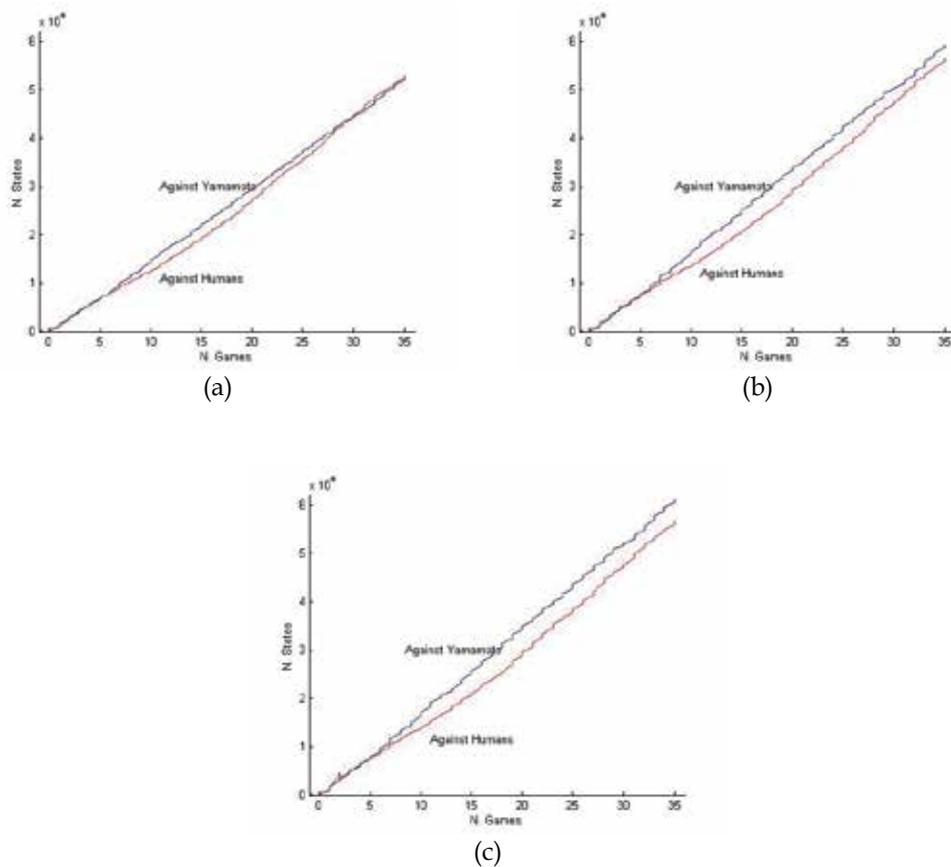


Figure 8. Number of states for roles (a) Goalkeeper, (b) Midfield, and (c) Striker

In figure 10 could be observed a comparison between the number of goals effectively made by both learners. In general, the team trained against humans achieved greater quantity of goals that the team trained against the Yamamoto's static strategy.

Although the learning process of this approach is too slow, it is important to say that learning by observation and analysis of visual information was (by our knowledge) first explored and was feasible to be implemented in robot soccer matches of robots.

7. Conclusion

One way to learn to make decisions using artificial intelligence in robotics is by leaving the learning for the rest time of the robotic system. This means, run the algorithms of learning in batches. Also, when talking about reinforcement learning in multi-robot and multi-agent systems, the paradigm proposed by authors showed better results than the traditional paradigms on the problems chosen for testing. Since these results encourage to continue this research using the model proposed in new problems and comparing it with previous proposals.

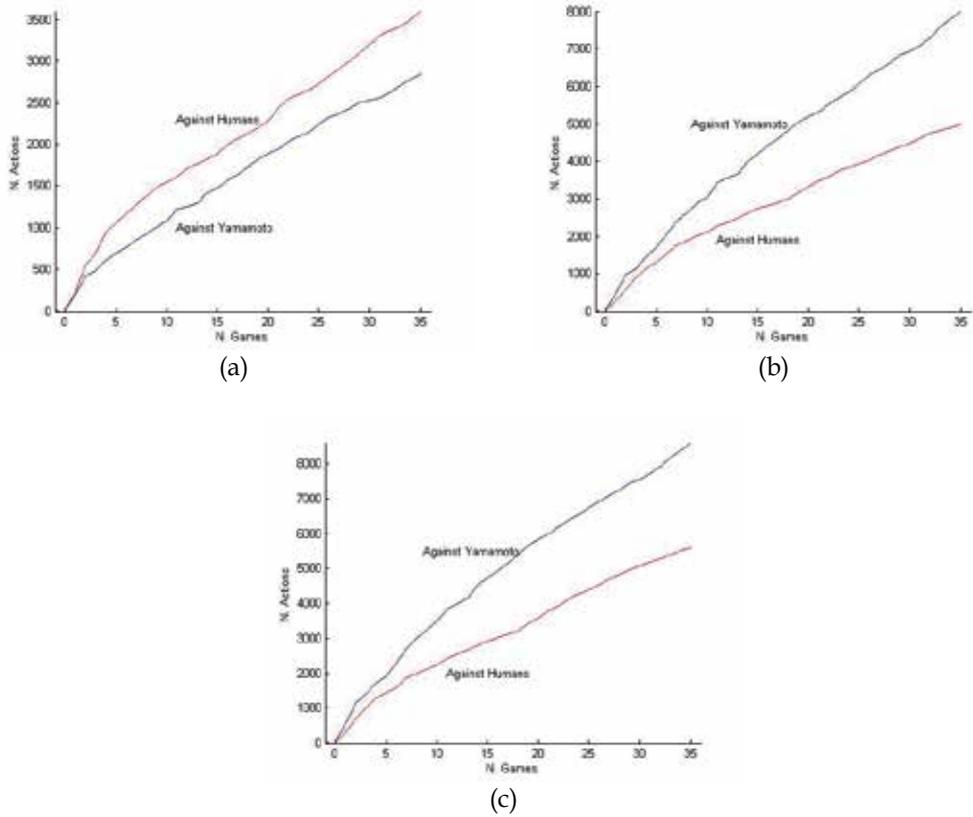


Figure 9. Number of actions for roles (a) Goalkeeper, (b) Midfield, and (c) Striker

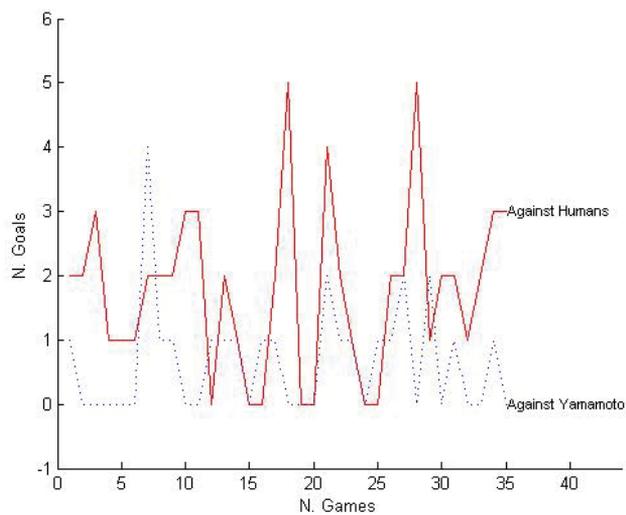


Figure 10. Number of goals effectively made by learners using this approach

During this work it was observed that learning by imitation is an appropriate technique to assist learning of techniques used in artificial intelligence, such as reinforcement learning. Thus, it is possible to overcome the limitations of these techniques when applied independently in complex problems, such as robot soccer. Limitations could appear either by the large number of states or by the large amount of available actions. Thus, learning by imitation can be used as a basis to carry out reinforcement learning in this kind of problems.

8. References

- Arai, T. and Ota, J. (1992). Motion planning of multiple mobile robots. In Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1992, volume 3, pages 1761-1768.
- Banerjee, D. and Sen, S. (2007). Reaching pareto-optimality in prisoner's dilemma using conditional joint action learning. *Autonomous Agents and Multi-Agent Systems* 15(1), 91-108.
- Barrios-Aranibar, D. and Alsina, P. J. (2007). Imitation learning: An application in a micro robot soccer game. *Studies in Computational Intelligence series, Mobile Robots: The Evolutionary Approach*. Volume 50, 2007, Pages 201-219 .
- Barrios-Aranibar, D. and Gonçalves, L. M. G. (2007a). Learning Coordination in Multi-Agent Systems using Influence Value Reinforcement Learning. In: 7th International Conference on Intelligent Systems Design and Applications (ISDA 07), 2007, Rio de Janeiro. Pages: 471-478.
- Barrios-Aranibar, D. and Gonçalves, L. M. G. (2007b). Learning to Reach Optimal Equilibrium by Influence of Other Agents Opinion. In: Hybrid Intelligent Systems, 2007. HIS 2007. 7th International Conference on, 2007, Kaiserslautern. pp. 198-203.
- Botelho, S. and Alami, R. (1999). M+: A scheme for multi-robot cooperation through negotiated task allocation and achievement. In Proceedings of 1999 IEEE International Conference on Robotics and Automation ICRA, pp. 1234-1239.
- Botelho, S. and Alami, R. (2000). Robots that cooperatively enhance their plans, Proc. of 5th International Symposium on Distributed Autonomous Robotic Systems (DAR 2000). Lecture Notes in Computer Science, Springer Verlag.
- Bruce, J., Bowling, M., Browning, B. and Veloso, M. (2003). Multi-robot team response to a multi-robot opponent team. In IEEE International Conference on Robotics and Automation, 2003. Proceedings. ICRA '03, volume 2, pages 2281 - 2286, Taipei, Taiwan.
- Chen, K.-Y. and Liu, A. (2002). A design method for incorporating multidisciplinary requirements for developing a robot soccer player. In Fourth international Symposium on Multimedia Software Engineering, 2002. Proceedings, pages 25-32, New-port Beach, California, USA.
- Claus, C. and Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. In Proceedings of the 15th National Conference on Artificial Intelligence -AAAI-98. AAAI Press., Menlo Park, CA, pp. 746-752.
- Filar, J. and Vrieze, K. (1997). *Competitive Markov Decision Processes*. Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY 10010, USA.

- Goldman, C. V. and Rosenschein, J. S. (1996). Mutually supervised learning in multiagent systems. In G.Weiß and S.Sen, eds., *Adaptation and Learning in Multi-Agent Systems*. Springer-Verlag: Heidelberg, Germany, Berlin, pp. 85–96.
- Guo, R., Wu, M., Peng, J., Peng, J. and Cao, W. (2007). New q learning algorithm for multi-agent systems, *Zidonghua Xuebao/Acta Automatica Sinica*, 33(4), 367–372.
- Iba, H. (1999). Evolving multiple agents by genetic programming. In U.-M.L. Spector, W. Langdom & P.Angeline, eds., *Advances in Genetic Programming*. Vol. 3, The MIT Press, Cambridge, MA, pp. 447–466.
- Jars, I., Kabachi, N. and Lamure, M. (2004). Proposal for a vygotsky's theory based approach for learning in MAS. In AOTP: The AAAI-04 Workshop on Agent Organizations: Theory and Practice. San Jose, California. <http://www.cs.uu.nl/virginia/aotp/papers/AOTP04IJars.Pdf>.
- Jolly, K. G., Ravindran, K. P., Vijayakumar, R. and Kumar, R. S. (2007). Intelligent decision making in multi-agent robot soccer system through compounded artificial neural networks. *Robotics and Autonomous Systems*. Volume 55, Issue 7, 31 July 2007, Pages 589–596.
- Kapetanakis, S. and Kudenko, D. (2002). Reinforcement learning of coordination in cooperative multi-agent systems. In *Proceedings of the National Conference on Artificial Intelligence*. pp. 326–331.
- Kapetanakis, S. and Kudenko, D. (2004). Reinforcement learning of coordination in heterogeneous cooperative multi-agent systems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2004*. Vol. 3, pp. 1258–1259.
- Kapetanakis, S., Kudenko, D. and Strens, M. J. A. (2003). Reinforcement learning approaches to coordination in cooperative multi-agent systems. *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)* 2636, 18–32.
- Kim, H.-S., Shim, H.-S., Jung, M.-J., and Kim, J.-H. (1997a). Action selection mechanism for soccer robot. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation, 1997. CIRA'97.*, Proceedings, pages 390–395.
- Kim, J.-H., Shim, H.-S., Kim, H.-S., Jung, M.-J., and Vadakkepat, P. (1997b). Action selection and strategies in robot soccer systems. In *Proceedings of the 40th Midwest Symposium on Circuits and Systems, 1997*, volume 1, pages 518–521.
- Kok, J. R. and Vlassis, N. (2004). Sparse cooperative q-learning. In *Proceedings of the twenty-first international conference on Machine Learning*. Banff, Alberta, Canada, p. 61.
- Kononen, V. (2004). Asymmetric multiagent reinforcement learning. *Web Intelligence and Agent System* 2(2), 105 – 121.
- Kuniyoshi, Y. and Inoue, H. (1993). Qualitative recognition of ongoing human action sequences. In *IJCAI93 Proceedings*, Chambéry, France, pages 1600–1609, 1993.
- Le Pape, C. (1990). A combination of centralized and distributed methods for multi-agent planning and scheduling. In *Proceedings of 1990 IEEE International Conference on Robotics and Automation ICRA*, pp. 488–493.
- Li, X. and Parker, L. E. (2007). Sensor Analysis for Fault Detection in Tightly-Coupled Multi-Robot Team Tasks. In *2007 IEEE International Conference on Robotics and Automation*. Roma, Italy, 10-14 April 2007, pp. 3269–3276.

- Miyamoto, H. and Kawato, M. (1998). A tennis serve and upswing learning robot based on bidirectional theory. *Neural Networks*, 11:1331-1344, 1998.
- Noreils, F. R. (1993). Toward a robot architecture integrating cooperation between mobile robots: Application to indoor environment, *The International Journal of Robotics Research* 12(2): 79-98.
- Oliveira, D. De and Bazzan, A. L. C. (2006). Traffic lights control with adaptive group formation based on swarm intelligence. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 4150 LNCS, 520-521.
- Panait, L. and Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*. 11(3), 387-434.
- Salustowicz, R. P., Wiering, M. A. and Schmidhuber, J. (1998). Learning team strategies: Soccer case studies. *Machine Learning*, 33(2): 263-282.
- Schaal, S. (1999). Is imitation learning the route to humanoid robots?. *Trends in Cognitive Sciences*, 3(6):233-242, 6 1999.
- Sen, S. and Sekaran, M. (1996). Multiagent coordination with learning classifier systems. In G.Weiß & S.Sen, eds., *Proceedings of the IJCAI Workshop on Adaption and Learning in Multi-Agent Systems*. Vol. 1042, Springer Verlag, pp. 218-233.
- Sen, S., Sekaran, M. and Hale, J. (1994). Learning to coordinate without sharing information. In *Proceedings of the National Conference on Artificial Intelligence*. Vol. 1, pp. 426-431.
- Shoham, Y., Powers, R. and Grenager, T. (2007). If multi-agent learning is the answer, what is the question?. *Artificial Intelligence*. 171(7), 365-377.
- Śnieżyński, B. and Koźlak, J. (2006). Learning in a multi-agent system as a mean for effective resource management. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 3993 LNCS - III, 703-710.
- Suematsu, N. and Hayashi, A. (2002), A multiagent reinforcement learning algorithm using extended optimal response. In *Proceedings of the International Conference on Autonomous Agents*. number 2, pp. 370-377.
- Sutton, R. and Barto, A. (1998), *Reinforcement learning: an introduction*. MIT Press, Cambridge, MA, 1998.
- Tumer, K., Agogino, A. K. and Wolpert, D. H. (2002). Learning sequences of actions in collectives of autonomous agents. In *Proceedings of the International Conference on Autonomous Agents*. número 2, pp. 378-385.
- Ulam, P., Endo, Y., Wagner, A. and Arkin, R. (2007). Integrated mission specification and task allocation for robot teams - design and implementation. In *2007 IEEE International Conference on Robotics and Automation*. Roma, Italy, 10-14 April 2007, pages 4428-4435.
- Wang, J. and Gasser, L. (2002). Mutual online concept learning for multiple agents. In *Proceedings of the International Conference on Autonomous Agents*. (2), 362-369.
- Wiering, M., Salustowicz, R. and Schmidhuber, J. (1999). Reinforcement learning soccer teams with incomplete world models. *Autonomous Robots*, 7(1): 77-88.
- Wu, C.-J. and Lee, T.-L. (2004). A fuzzy mechanism for action selection of soccer robots. *Journal of Intelligent and Robotic Systems*, 39(1): 57-70.

- Yamamoto, M. M. (2005). Planejamento cooperativo de tarefas em um ambiente de futebol de robôs. Master's thesis in electrical engineering. Federal University of Rio Grande do Norte. Brazil.
- Zhang, Y. and Mackworth, A. K. (2002). A constraint-based robotic soccer team. *Constraints*, 7(1): 7-28.

Cellular Non-linear Networks as a New Paradigm for Evolutionary Robotics

Eleonora Bilotta and Pietro Pantano

*Università della Calabria
Italy*

1. Introduction

One of the most active fields of research in evolutionary robotics is the development of autonomous robots with the ability to interact with the physical world and to communicate with each other, in “robot societies”. Interactions may involve a range of different motor actions, motivational forces and cognitive processes. Actions, in turn directly affect the agent’s perceptions of the world. In the “Action/Perception-Cycle” (see Figure 1), biological organisms are integrated sensorimotor systems. This means that intelligent processes require a body, and that symbols are grounded in the environment in which animals live (Harnad, 1990). In short, behavior is fundamentally linked to cognition. This is true for humans, animals and artificial agents. Without this grounding, artificial animals and agents cannot live and behave successfully in their artificial environments. One way of achieving it, is to use Genetic Algorithms to evolve agents’ neural architecture (Nolfi & Floreano, 2000). This creates the prospect of robots that can live in complex socially organized communities in which they communicate with humans and with each other (Cangelosi e Parisi, 2002). According to these authors, cognition is an intrinsically embedded phenomenon in which the dynamical relations between the neural system, the body and the environment play a central role. In this view, agents are dynamical systems and cognitive functioning has to be understood using tools from dynamical system theory (van Gelder, 1995, 1998a; 1998b; Bilotta et al., 2007a-2007f). This perspective on cognition has been called the Dynamical and Embodied view of Cognition (DEC) (Keijzer, 2002).

In this chapter we describe our own contribution to Evolutionary Robotics, namely a proposal for a new generation of believable agents capable of life-like intelligent communicative and emotional behavior. We focus on CNNs (Cellular Neural Networks) and on the use of these networks as robot controllers. In previous work, we used Genetic Algorithms (GAs) to evolve Artificial Non-linear Networks (ANNs) displaying artificial adaptive behavior, with features similar to those observed in animals and humans. In (Bilotta et al. 2006), we replaced ANNs with a new class of dynamical system called Cellular Non-linear Networks (CNNs) and used CNNs to implement a multilayer locomotion model for six-legged artificial robots in a virtual environment with many of the characteristics of a physical environment. First invented by Chua and co-workers (1988), CNNs have been extended to create a CNN Universal Machine (CNN-UM) (Roska & Chua, 1993), the first algorithmically programmable analog computer chip suitable for the modeling of sensory-

motor processes. Applications of the chip include the modeling of the mammalian retina (Roska et al. , 2006) and motor coordination in life-like robots (Arena & Fortuna, 2002). CNNs can be organized in complex architectures of one, two or three-dimensional processor arrays in which the cells are identical non-linear dynamical systems, whose only connections are local. Systems composed of CNNs share a large number of features with living organisms: local connectivity and activity, nonlinearity and delayed synapses for processing tasks, the role of feedback in influencing behavior, as well as combined analog and logical signal-processing. Direct implementation on silicon provides robust, economic, fast and powerful computation. Thousands of experiments have demonstrated the possibility of digitally programming analog dynamics. This has made the CNN paradigm into a useful tool for robot applications.

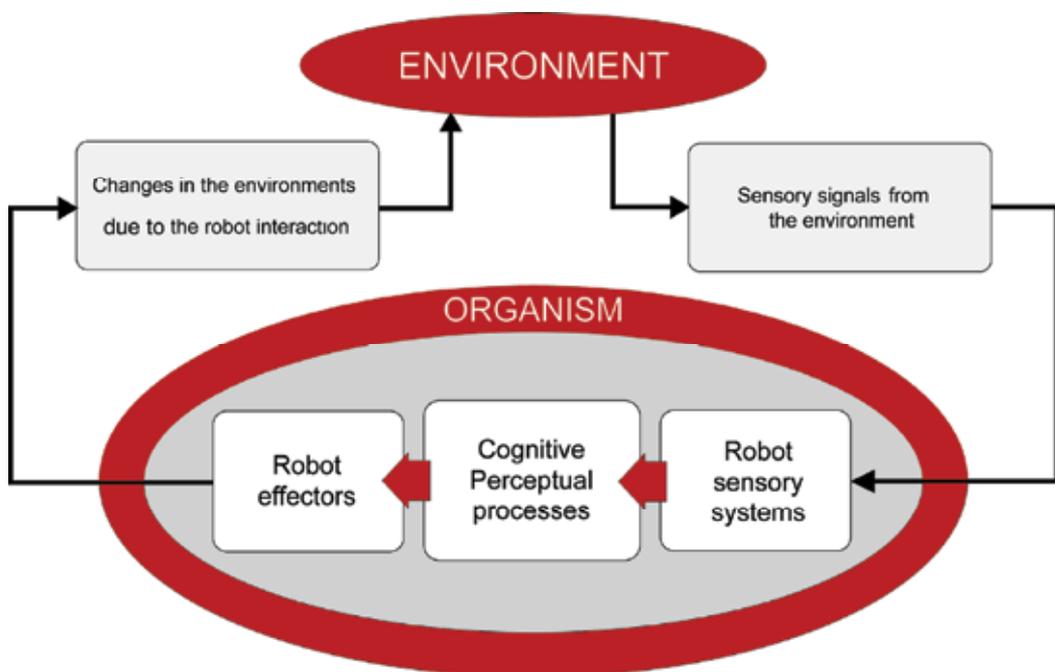


Figure 1. The Action/Perception-Cycle

In this chapter we will proceed as follows. In Section 2, we introduce the concept of Cellular Non-linear Networks as innovative dynamical robot controllers that can be implemented both in simulation and as hardware prototypes. In section 3 we present CNN architectures for different cognitive and motor processes (CNN computing: visual and motor modalities); in section 4 we present the RoVen simulation environment – an environment specifically developed for the evolution of CNN-based robot controllers. In Section 5 we describe our experiments in simulated environments. Section 6 describes the implementation of prototype chips which can act as behavioral modules for physical robots. In the conclusions, we summarize the evidence that the CNN paradigm can dramatically improve current research in Evolutionary Robotics.

2. Cellular Neural Networks

CNNs (Cellular Neural Networks) were first introduced in 1988 by Leon Chua and Yang (Chua & Yang, 1988). They are dynamical systems. For this reason they are sometimes referred to as Cellular Nonlinear Networks. CNNs have applications in many domains from image recognition to robot control. Given their ductility, the ease with which they can be implemented and the dynamics they display they can be considered a *paradigm for complexity*. CNNs can be organized in one- two- or three dimensional topologies (see Figure 2). As we will see in the following sections, CNN applications are of relevance to many different disciplines including Robotics, Dynamic Systems Theory, Neuro-psychology, Biology and Information Processing. One of the first applications was *image processing*. A digitalized image can be represented as a two dimensional matrix of pixels. To process it with a CNN, all that is necessary is to use the normalized colors of pixels (i,j) as the initial state of the network. The network then behaves as a non-linear dynamical system with a number of equations equal to the number of cells. The network makes it possible to perform a number of useful operations on the image. These include edge detection, generation of the inverse figure etc. For additional information on this topic and on other applications of CNNs, see (Chua, 1998).

As we will see in Section 6, CNNs are very similar to programmable non-linear dynamical circuits – and in fact physical implementations often use these circuits. Given CNN's non-linear design, CNNs often produce *chaotic dynamics*. Given the presence of *local activity* in individual cells, it is possible to observe a broad range of emergent behaviors. One of these is the formation of Turing patterns (Chua, 1995), which are often used in robot control. (Arena et al., 1998; Arena & Fortuna 2002).

As with all complex emergent phenomena, it is difficult to identify the full range of non-linear dynamic behaviors a CNN can produce and equally hard to control the network's behavior. The main reason is that the dynamics of individual cells are controlled by first order non-linear differential equations. Given that the cells are coupled, the equations are also coupled. This makes them similar to the Lorenz system and Chua's circuit (Bilotta et al. 2007a-2007f) which also display highly complex, mathematically intractable dynamics. What is special about CNNs is that they can be used to reproduce the complex dynamics of other non-linear systems such as Chua's circuit (Bilotta et al., 2007a). In this sense, we can consider CNNs as a *general model* or a *meta-model* for other dynamical systems.

Another important application of CNN is in the numerical solution of Partial Differential Equations (PDEs). If we use a grid to create a discretized space, in which variable values are represented by intersections on the grid, the derivatives with respect to spatial variables can also be discretized while the derivatives with respect to time remain unchanged. These discretized differential equations can be mapped onto the equations regulating the behavior of the CNN. In this way, CNNs can simulate a broad range of physical phenomena. For more information and a review of this aspect of CNN see (Chua, 1998).

Specific CNN architectures can reproduce a broad range of *non-linear phenomena* that biologists, neurologists and physics have observed in active non-linear media and in living tissue. These include solitons, eigen waves, spiral waves, simple patterns, Turing patterns etc. Given CNN's local connectivity, diffusion is a natural property of the network; diffusion-reaction dynamics can be simulated using the interactions between an inhibitory and an excitory layer. This class of two-layer CNN has been called a Reaction-Diffusion CNN.

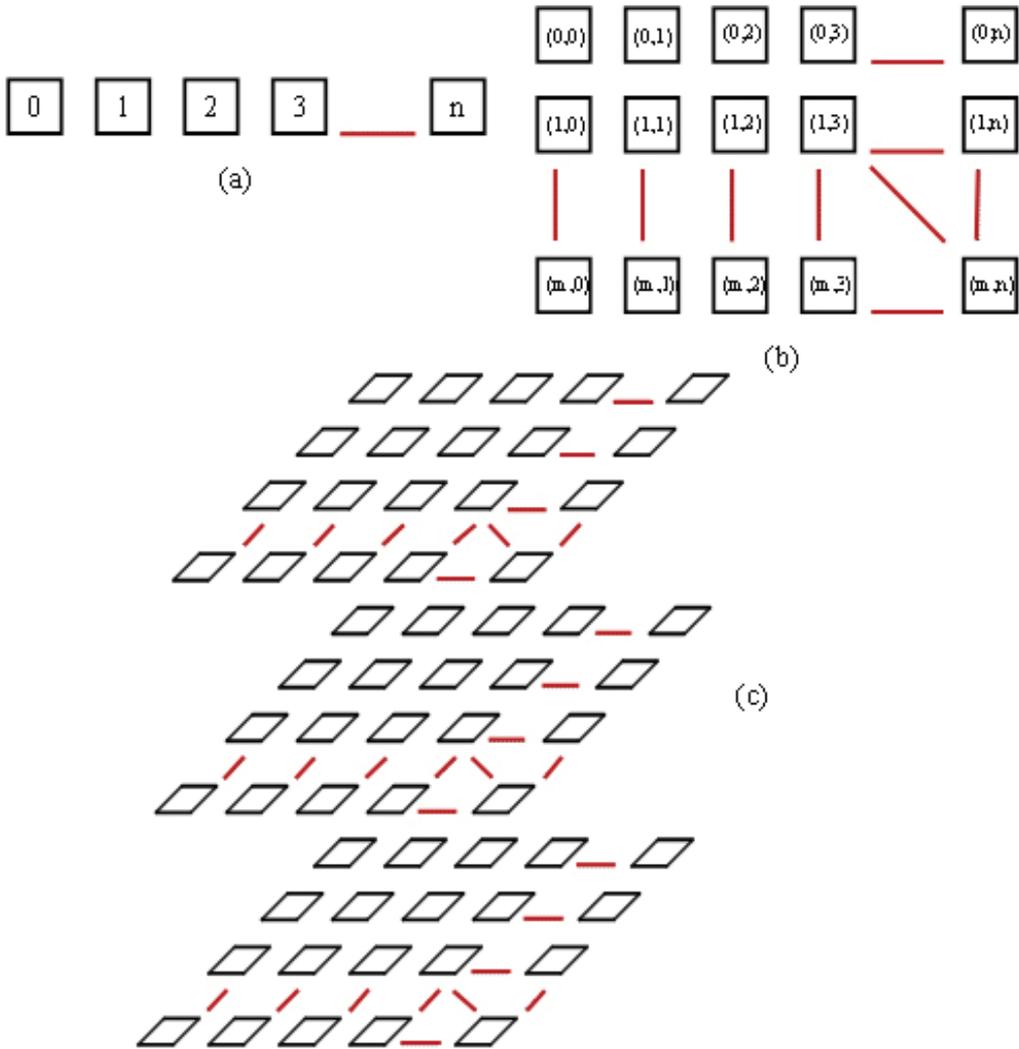


Figure 2. CNN topologies a) A linear topology; b) A two-dimensional topology; c) A three-dimensional topology

From a mathematical point of view a CNN is a discrete set of continuous dynamical variables called “cells”. Each cell is associated with three independent variables: the input, the threshold and the initial state. The cell’s dynamics are influenced by close-by cells $S_{ij}(r)$ in a neighborhood of radius r . Thus, if we consider a CNN with dimensions $L \times M$, the dynamics of cell C_{ij} , located on the i -th row and the j -th column, are influenced by cells in the neighborhood $S_{ij}(r)$ defined as:

$$S_{ij}(r) = \{C_{ij}, with 1 \leq i \leq L; 1 \leq j \leq M\} \tag{1}$$

that is the set of cells lying within a sphere of radius r , centered on C_{ij} .

The standard equation for CNNs (Chua, 1988) is thus:

$$\begin{aligned}\dot{x}_{ij}(t) &= -x_{ij} + \sum_{kl \in S_{ij}(r)} a_{kl} y_{kl} + \sum_{kl \in S_{ij}(r)} b_{kl} u_{kl} + z_{ij} \\ y_{ij}(t) &= f(x_{ij}) = \frac{1}{2} (|x_{ij} + 1| - |x_{ij} - 1|) \\ i &= 1, 2, \dots, L, j = 1, 2, \dots, M\end{aligned}$$

where x_{ij} , y_{ij} , u_{ij} and z_{ij} are the scalar functions *state*, *output*, *input* and *threshold* for cell C_{ij} ; a_{kl} , and b_{kl} are additional scalar functions which we will refer to as *synaptic weights*.

A CNN with $r=1$, can be identified by just 19 real numbers (a uniform *threshold* $z_{ij} = z$, 9 synaptic weights for feedback a_{kl} , and 9 synaptic weights for control b_{kl}). These 19 numbers are the CNN's "genes". The set of all CNN genes constitute the CNN *genome*.

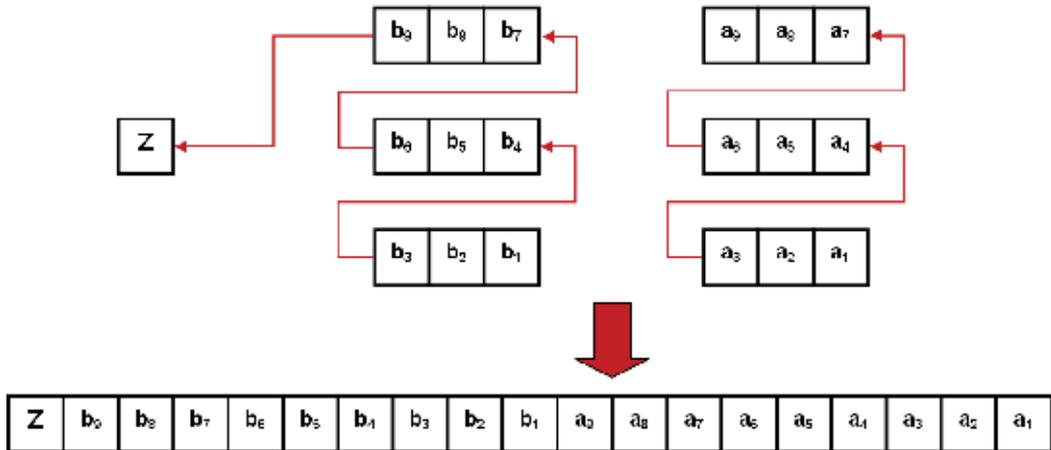


Figure 3. Constructing the genes of a CNN

As already mentioned, there are many applications where the most useful CNN is the network shown in Figure 4. In *two-layer* CNNs each cell is double. Alternatively we could say that each cell has two state variables. We can thus distinguish between C_{ij}^1 (cells in layer 1) and C_{ij}^2 (cells in layer 2). In this kind of two layer network the neighborhood of a cell is defined by:

$$\begin{aligned}N_{r_1}(i, j) &= \{C_{ij}^1, \text{with } 1 \leq i \leq L; 1 \leq j \leq M\} \\ N_{r_2}(i, j) &= \{C_{ij}^2, \text{with } 1 \leq i \leq L; 1 \leq j \leq M\}\end{aligned}$$

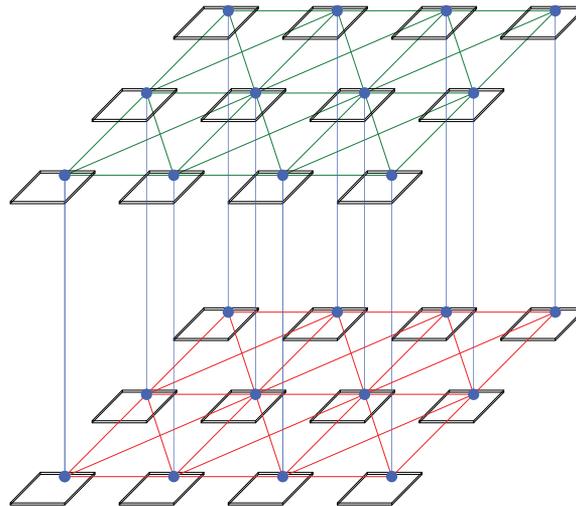


Figure 4. An example of a two layer planar architecture showing connections between cells in the same layer (green in layer 1, red in layer 2) and between twin cells in different layers (shown in blue)

Thus we obtain one definition of neighborhood for cells belonging to layer 1 and a second definition for cells belonging to layer 2. Cells intercommunicate only if they have the same index.

3. CNNs Architectures

We have already seen that CNNs can be useful in a broad range of applications. Now we will focus on three specific applications in robotics:

- a) Models of the Central Pattern Generator (CPG);
- b) Applications in artificial vision and new models of artificial retinas.

3.1 Central Pattern Generator

One of the central problems in bio-inspired robotics is to reproduce animal locomotion in artificial settings. Experiments with simple invertebrates have helped researchers to understand and model the anatomical and functional mechanisms underlying animal locomotion (Arena & Fortuna, 2002; Schilling et al., 2007). In animals with very simple locomotion (such as certain worms and molluscs), movement depends on direct propagation of a signal through nerves. The soft structure of the animal's body allows it to synchronize with the waves and to produce a wave-like locomotion. This kind of phenomenon can be easily described by the reaction-diffusion PDEs mentioned earlier (Murray, 1989). As already stated, the solutions to the equations include autonomous oscillations with all the properties of an eigen-wave (Krinsky, 1984). The same mechanism – in which continuous motion is maintained by an eigen-wave – is also present in higher animals. However, the underlying neural structure is much more complex and shows a much higher degree of organization. To model these behaviors mathematically we can begin by studying the various components of the animal's nervous system. The resulting model is built around a Central Pattern Generator (CPG) in which motor neurons are activated by

stimuli from the Central Nervous System (CNS). The neural structure of the CPG is very complex. As a result, it can do more than just generate the impulses controlling movement; it can also control the transition from one kind of movement to another (Pearson, 1993). In one possible model (see Figure 5) the CPG is described as a complex, hierarchically organized excitatory-inhibitory system. Within this system a group of control neurons (CNs) receives stimuli from the Central Nervous System and activates other neurons (LPGN) that generate timing signals appropriate to the desired form of locomotion (Calabrese, 1995).

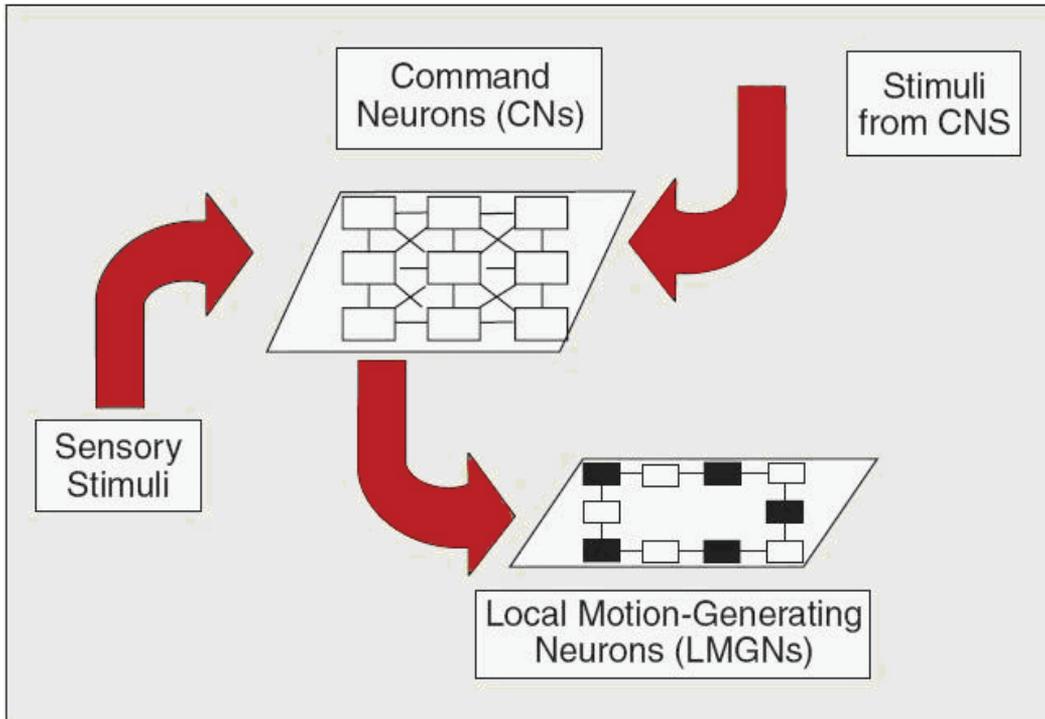


Figure 5. A schematic diagram of the CPG from (Arena & Fortuna, 2002)

This model can be easily represented by a two layer CNN, based on the reaction-diffusion equations mentioned earlier. The use of these equations allows it to generate spatial-temporal patterns such as eigen-waves and Turing patterns. This means we can use CNN both to model the CPG and to control robot locomotion. Further details are available in (Arena & Fortuna, 2002; Schilling et al., 2007).

3.2 Vision and Artificial Retinas

CNNs can be used to simulate higher level cognitive processes and are beginning to be used as models of complex processes relevant to the construction of artificial organs. One particularly important example is the retina. In a series of experiments in rabbits, Werblin and coworkers (Fried et al., 2005)-(Roska et al. 2006), have demonstrated that the retina pre-processes visual signals before sending them to the brain. More specifically, they show that before being sent to the cortex for final processing the retina uses excitatory and inhibitory mechanisms to break it down into 12 separate sub-images. These are processed in parallel each with its own clock. The result is a seamless flow of visual information. On the basis of

this work, the authors went so far as to propose the idea of an *alphabet for vision* – and argued that understanding this natural language was one of the most important problems in modern science. It is a problem with great relevance to the construction of artificial organs. As Werblin and Roska pointed out (Roska et al., 2006), it is extremely difficult to connect electrodes to individual retinal cells in the living rabbit. This makes it hard to understand how the rabbit responds to visual stimuli e.g. a one second flash or a minute's exposure to a natural scene. For their experiments and the analysis of the resulting data, they used a model based on CNNs. The results were astonishing and showed that a CNN can reproduce many phenomena observed in vivo. Obviously this modeling effort is still in its initial stages. Nonetheless it provides evidence that CNNs can be very useful in modeling complex natural phenomena such as those observed in the retina. Remember that CNNs are based on PDEs. With their emergent data processing properties, their parallel processing capabilities and their ability to process continuous flows of information they represent an important paradigm in modern complexity science.

4. Evolving CNNs with GAs

In the previous section we saw how we can use CNNs to model a range of processes (e.g. locomotion control, retinal image processing) that are highly relevant to robotics. This suggests they could act as a unifying model for a new generation of bio-inspired autonomous robots. The basic schema shown in Figure 6 can be used as a basic design both for physical and simulated robots.

To implement the schema, we intend to develop experimental scenarios in which simulated robots are controlled by dynamical systems which allow them to achieve the kinds of perceptual-motor, communication and emotional behavior they need to interact effectively with other robots and with humans. In these scenarios, our robots will demonstrate a high degree of autonomy and self-awareness.

It is obvious that implementing the architecture just described is a highly complex task which must necessarily be decomposed into subtasks. These include:

- a. implementation of a sensory visuomotor architecture, allowing them to merge information in different sensory modalities into a coherent representation of the environment. By using CNNs, we intend to endow robots with artificial visual, auditory and haptic systems allowing them to recognize and categorize faces, objects, and scenes under varying viewing and dynamically changing environmental conditions (Roska & Chua, 1993; Gacsádi et al., 2006; Gacsádi and Szolgay, 2004);
- b. implementation of an emotional architecture, organizing the robot's behavior. An emotional robot would use its physical actuators and cognitive skills to adapt to changing environmental conditions in real-time. In this setting, emotions trigger behavior. More specifically, moving objects, the postures/gestures of other robots, and sounds or facial expressions produced by human beings all generate different emotional states;
- c. implementation of a communication system based both on non-verbal communication (Gesture and movement recognition, Face recognition) and a natural language model. Specific learning mechanisms will allow CNN- based devices to acquire, from human users the speech configurations of different natural languages. A specific CNN-based architecture, combined with sensors attached to a human speaker, will allow the system to capture the emotions expressed by human subjects when interacting with a virtual interface (Face and emotion recognition). CNNs will perform recognition tasks in two sensorial modalities (auditory and

visual). We will then use GAs to spontaneously evolve spoken language. In parallel with this work, we will implement a robot controller that will enhance the cognitive mechanisms the robot uses to process emotions thereby dramatically improving human-computer interaction; d. implementation emergent social behavior with hundreds of agents. We will use acoustic stimuli (known and unknown sounds) and visual stimuli to verify how agents' interactions with the external world influences the dynamics of large groups of robots and the way communication emerges within the group.

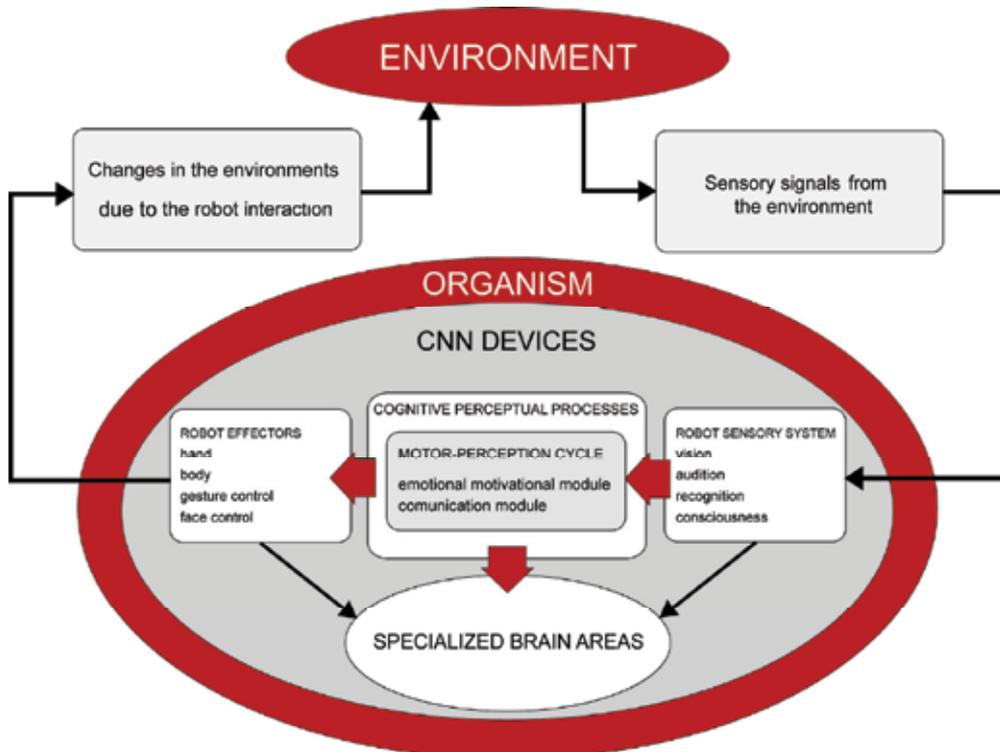


Figure 6. A cognitive architecture for CNN based Robots

The key features of our approach include:

1. Behavioral/cognitive modeling for robotics.
2. A dynamical approach to the modeling of robot cognition and emergent behavior.
3. Integration of mathematical models on different scales and at different levels of the physical functional architecture.
4. Robust behavioral hardware and software implementation with Cellular Neural Networks (CNNs), supporting parallel multidimensional processes that allow robots to robustly handle their interaction with the environment.
5. Robust implementation of robot controllers using Cellular Neural Networks (CNNs), evolved by evolutionary techniques (GAs).

It is obvious that what we are proposing will require a great deal of work and that at the moment we are only in the very early stages. So far we have developed an initial simulation environment that we have called RoVEn . RoVEn allows us to model robots, to implement robot controllers and to evolve them using genetic algorithms. In what follows we describe:

- a) The RoVEn (Robot Virtual Environment) simulation environment.
- b) A number of specific robots we have used in our experiments.
- c) Our “evolutionary laboratory”.

Any experiment in evolutionary robotics can be divided into the following steps:

1. Design and implement the robot body.
2. Insert the robot controller.
3. Associate the controller with a genome, susceptible to modification by genetic operators.
4. Analyze the evolutionary process.
5. Analyze the behavior of the “best behaving” robots using a simulation environment.

RoVEn makes it possible to perform all these steps in a single simulation environment. The environment was developed using Java 3D libraries for rendering and ODE (Open Dynamics Engine) libraries for the simulation of the physical world. The ODE simulation engine contains a numeric integrator that solves the equations of motion for inelastic bodies. Below we provide more details of the simulation environment.

4.1 Creating a robot body in the ROVEN simulation environment

The robot body is modeled as a set of inelastic bodies connected via joints. The use of joints makes it possible to define hierarchies of bodies and to create complex prototypes. Each body has a Shape, characterized by a color and geometry.

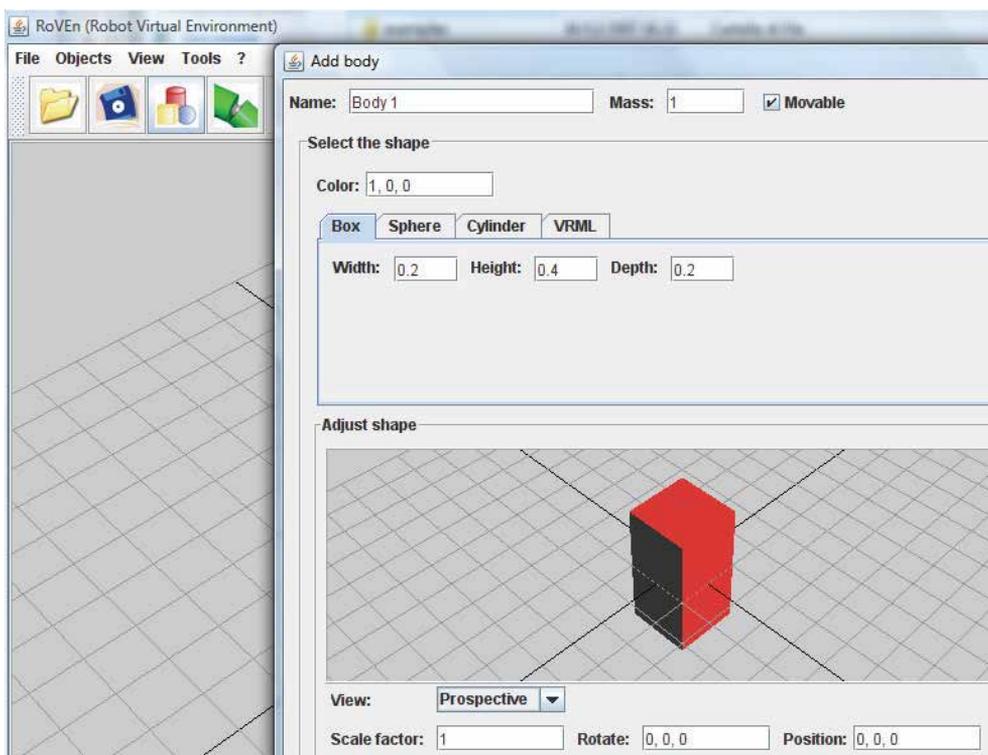


Figure 7. The RoVEn environment used to create a single inelastic body. The lower part of the window contains controls making it possible to move, rotate and change the scale of bodies

RoVEn uses three elementary geometries (see figure 7):

- a) Parallelograms.
- b) Spheres.
- c) Cylinders.

Obviously this is not enough to create complex shapes such as arms and legs. For this purpose it is possible to import additional shapes from VRML files. A Movable option makes it possible to distinguish between mobile objects (e.g. robots and robot parts) and immobile objects (a wall, an inclined plane etc.).

To join one body to another, the user selects the first body, clicks on the Join button on the Tool Bar and clicks on the second body. Any body which is jointed to another body (via a spherical joint) contains a motor, positioned on the pin on which the joint rotates. This is why it is not possible to have more than one motor per body. It is possible to define limits on how far the body can rotate along its three axes of rotation. Alternatively constraints can be inserted one at a time by setting the properties: Elevation bounds, Azimuth bounds, and Tilt bounds.

Figure 8 shows a six-legged agent created with RoVEn. It is this agent we used in the examples we refer to later. Figure 9 shows how we can use the same components to construct a humanoid or a four legged robot in the RoVEn environment.

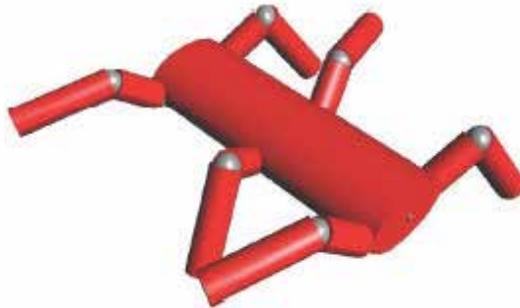


Figure 8. A six-legged body composed of a central body and 12 spherical joints

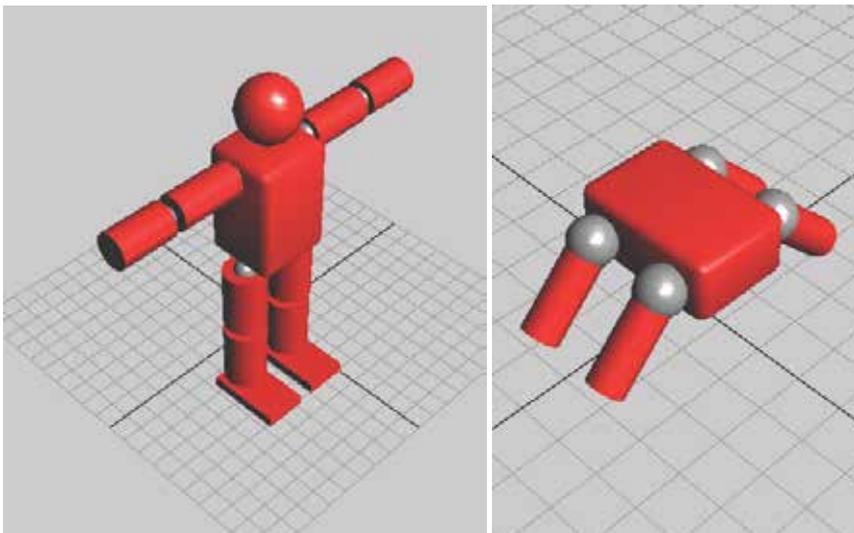


Figure 9. A four-legged and a humanoid robot in RoVEn

Actuators and sensors have to be installed on the bodies. To install a sensor, it is enough to select the body and click on *Add Sensor*, on the Tool Bar. This opens a window which can be used to install a new sensor. RoVen provides four classes of sensor: a clock sensor which cycles between an output of 1 and an output of 0 at predefined times, a position sensor, a proximity sensor and a WebCam.

4.2 Controllers

RoVen makes it possible to simulate different classes of controller and their interactions. *Controllers* read data from sensors and act on the actuators in such a way that the robot takes a specific action. Interacting controllers can read and write to special *registers* that simulate the registers normally provided by computer hardware. We have implemented several different classes of controller. Below we describe them briefly.

Interface

This class of controller provides an interface between devices and registers (which can be read and modified by other controllers). For each class of device, RoVen displays data describing components suitable for connection to registers. For instance, for motors installed on joints, the system displays three text boxes, showing the names of the registers where the system stores the robot's speed along the three axes of rotation and an additional three text boxes showing the registers with the angles reached by the motor. Alternatively the values can be inserted by hand. The proper location for interface controllers connected to sensors is a low layer of the controller which uses the interfacing registers. In this way, the controller can use up to date information for every interaction. Otherwise the values in the register would always refer to the previous interaction.

User control

This controller is used exclusively to control motors. When a simulation begins, the user uses a special window to define the speed of the motor along the three axes of rotations.

Time series

This controller allows the user to define how the speed of the motor changes over time. One method is to import a text file defining the speed of the motor along the three axes of rotation at specific times. Alternatively the user can set a mathematical function defining the speed of the motor as a function of time.

Chua circuit

This controller makes it possible to define a set of Chua circuits connected by resistors and to use the system. The current and voltage produced by this system controls a motor. A special control panel allows users to define the properties of the individual components.

CNN

These controls make it possible to emulate a CNN and to connect the CNN to the input and output registers, available to other controllers. The control panel contains a number of different tabs (see Figure 10). The *Cell* tab allows the user to define the Feedback and Feed-forward matrices, the threshold for the cell and the size of the network. The *Input* and *Output* tabs allow the user to define the registers where the network has to read its input and write its output. The *Option* tab allows the user to set the parameters for the CNN, including the integration step (in the *Delta* field) and the boundary conditions (see Figure 10).

The controller is designed to support the applications described in Section 3. As we will see in the following section, the values of the Feedback e Feed-forward matrices can be optimized using evolutionary techniques (genetic algorithms). The network can take input

from several different kinds of sensor. In the experiment described in the following section, we use a position sensor to evaluate the trajectory followed by the robot.

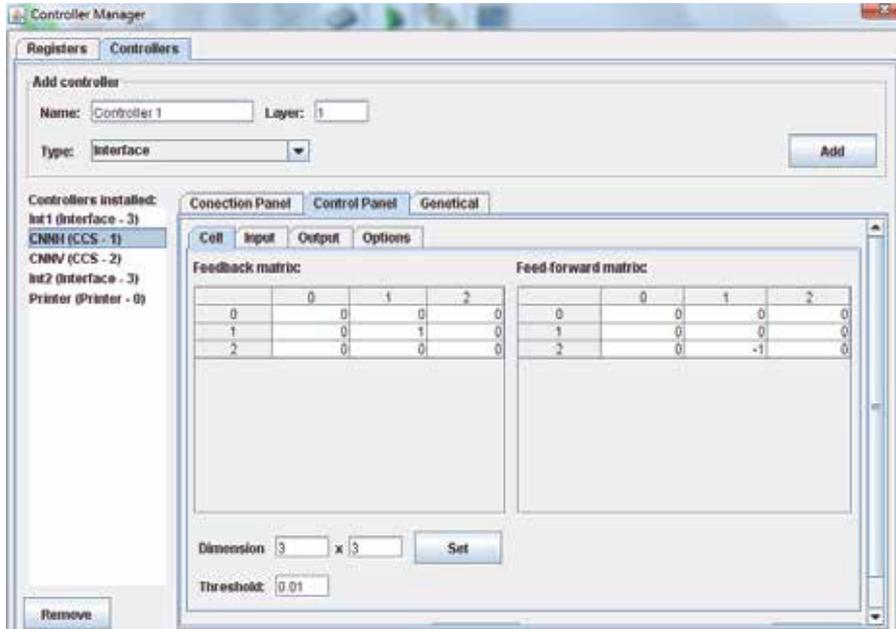


Figure 10. The CNN Control Panel

Neural Networks

In addition to the controllers mentioned earlier, the system also includes a neural network controller. This makes it possible to replicate classical experiments in evolutionary robotics. For further details see “Evolutionary Robotics” (Nolfi & Floreano, 2000).

Other controllers

In cases in which existing controllers are not sufficient, RoVen allows users to develop their own *ad hoc* controllers. Ad hoc controllers are developed in **Java** and loaded into RoVen as plug-ins. RoVen provides developers with a special SDK to help them in building plug-ins and in importing them into applications.

4.3 A genetics laboratory for CNN

Genetic Algorithms (GAs), invented by John Holland (Holland, 1993), are an optimization technique which has been applied to a large class of problems where classical techniques do not provide appropriate solutions. GAs are inspired by Darwin’s theory of evolution through natural selection. The general idea is to evolve problem solutions using techniques similar to those nature uses to evolve animal species.

Given a system that needs to perform a specific operation, GAs optimize the parameters of the system by evolving optimally performing individuals. Having specified the problem in this way, we can see GAs as a means to generate a desirable behavior. A necessary condition is that the system can be fully specified by a finite set of parameters. The parameters are then represented as a string of characters (the system’s “DNA” or genotype). In this setting, each parameter is a gene; the working system in which these genes are expressed is the “phenotype”. In the work we describe below the phenotype consists of a robot controlled

by a CNN; the genotype is the set of parameters controlling the CNN, that is the values of the feedback and feedforward matrices.

Select a controller to evolve: CNNH (CCS - 1)

Evolution parameters:

General Laboratory Task

Mutation probability: 0.03

Number of crossover points: 5

Fitness function: z

Threshold range: -1.0 : 1.0

Dimension: 3 x 3 fill with: -1, 1 Set

CNN mask:

Feedback matrix:

	0	1	2
0	-1, 1	-1, 1	-1, 1
1	-1, 1	-1, 1	-1, 1
2	-1, 1	-1, 1	-1, 1

Feed-forward matrix:

	0	1	2
0	-1, 1	-1, 1	-1, 1
1	-1, 1	-1, 1	-1, 1
2	-1, 1	-1, 1	-1, 1

Manipulate Manipulate

Figure 11. The window used to control genetic operators and initial values for the CNN controller

An important phase in the GA is the evaluation of the fitness of different phenotypes. In the work described here, we simulate the behavior of the robot in a simulator, extract behavioral indicators (e.g. Lagrangian parameter values and their derivatives, distance covered etc.) and use the values of these indicators as the arguments of a *fitness function* f that computes the fitness of the robot.

A Genetic Algorithm comprises the following steps:

1. Generation: generation of an initial population of individuals with randomly generated CNN parameters;
2. Fitness evaluation: simulation of the behavior of each individual robot and computation of fitness values for each individual;
3. Selection: selection of the individuals with the highest fitness value (a certain percentage of the population);
4. Reproduction: generation of new individuals by *crossing-over* the genotypes from two parent individuals and randomly *mutating* a certain percentage of their genes;
5. Go to step 2: repetition of the process through the creation of a new generation of individuals.

In defining a GA, one of the key steps is the choice of the fitness function. Different problems require different fitness functions adapted to their specific requirements. Another important issue is the optimization of the code so as to allow as many iterations of the algorithm as possible in the shortest possible time (Mitchell, 1996).

The “Evolutionary Laboratory” we used to evolve the CNN controller allowed us to define the following parameters:

- Mutation probability: the probability (a value between 0 and 1) that any given gene will undergo mutation in a single iteration of the algorithm.
- Number of cross-over points: the number of cross-over points.
- Evaluate fitness on axis: the axis along which to measure the distance traveled by the robot. The result of this measurement is used to evaluate the robot’s fitness.
- Threshold range: the range of possible values for the Threshold parameter.

Figure 11 shows one of the user interfaces used during the evolution of the CNN controller for the robot

4.4 Analysis of evolutionary trends

The output of the GA consists of the genotypes present in the last generation of individuals produced by the algorithm. The file containing these values can be loaded using the *Genetical Tab* in the *Processor Manager* window. The *Load* button allows the user to load the files for all individuals with fitness higher than a user-specified threshold. The user can then select an individual and use the *Set* button to copy its DNA to the current controller.

An additional Data Analysis module allows the user to display a graph showing mean and maximum fitness values at each step in the evolutionary process.

4.5 Simulating the behavior of optimal individuals

The final environment allows the user to observe the behavior of optimal individuals. (see Figure 12). A printer interface makes it possible to print the results of the simulation to a file where they can then be analyzed using spreadsheet software.

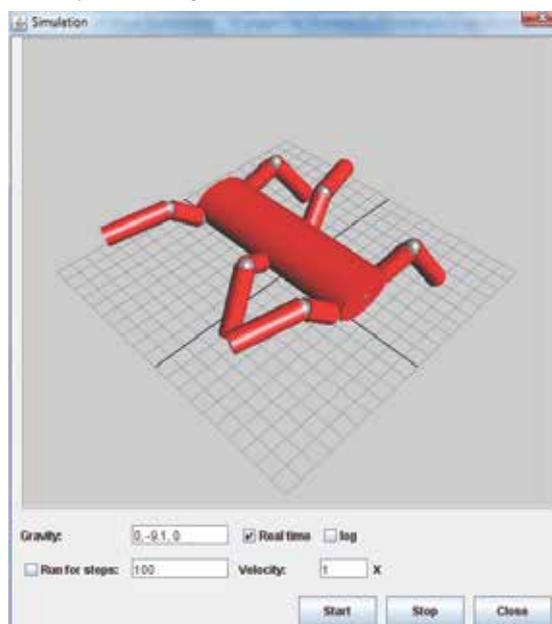


Figure 12. The behavior simulation window

5. Experimental Setting

We performed a number of simulations, using a range of different controllers. In what follows we present an example from this work.

5.1 Robot design

The robot (see figure 13) consists of a body, connected to six legs. Each leg is made up of two parts. Where parts are connected, the connection is a spherical joint. In the case of legs, one “internal joint” joins the leg to the body; a second “external joint” joins the second part of the leg to the first.

In what follows we will consider a configuration in which the internal joint is restricted to movement along the azimuth. In this way the internal joint can only move ahead of and behind the legs. We also eliminate two degrees of freedom for the external joint, restricting it to elevation rotation (raising and lowering the second part of the legs). In this way, if we know the position of the robot’s center of gravity on the plane, and the direction in which it is moving, the system has just 12 degrees of freedom (the two angles of rotation associated with each of its six legs). (If we assume that the body is always in contact with the ground it has 15 degrees of freedom). The movement of the robot’s legs, starting with this initial configuration, uniquely determines the robot’s position. Elevations are constrained within the range 0° to 45° ; the azimuth is constrained to the range 20° to $+20^\circ$. This prevents the legs from touching and creating problems for the simulator.

For the robot controller, we used two CNNs connected to the internal joints, and one connected to the external joints. Since the robot has six legs, there are $6 + 6$ hinges, each with a motor connected to a cell in the CNN. The output from each cell provides the input for the motors, as shown in Figure 13.

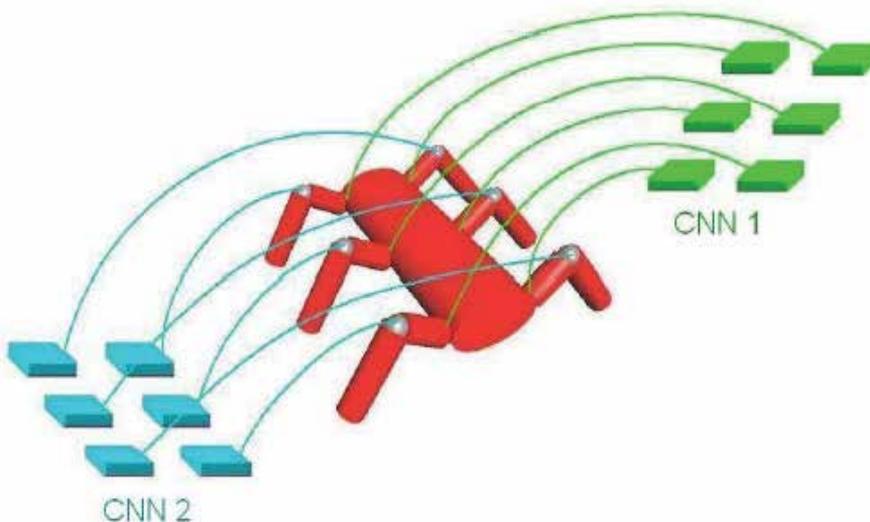


Figure 13. The robot controller. The output from 2 6-cell CNNs is directly connected to the motors

Each CNN consists of 6 cells, each cell lies in a neighborhood with $r=1$. Thus each cell has 9 neighbors (including itself). Given that all cells have a neighborhood of the same size, the boundary conditions for the network will be periodical (see Figure 14).

The output from each cell defines the input to the motors as shown in Table 1

CNN1		CNN2	
IFLJ	IFRJ	EFLJ	EFRJ
IMLJ	IMRJ	EMLJ	EMRJ
IBLJ	IBRJ	EBLJ	EBRJ

Table 1. Connections between the outputs of the two CNNs and the 12 motors

To identify the motors we use a code based on the following conventions. The first letter in the code indicates whether the joint is internal (I) or external (E); the second indicates whether the joint is on a front, middle or back leg (F, M, B). The third letter shows whether the joint is on the left (L) or the right (R). The last letter (J) indicates that the code refers to a joint. Each cell in the first CNN takes its input from the angular sensors inside the hinge on the joint to which it delivers its output. The second CNN takes its input from the first (the first cell of CNN 1 is connected to the first cell of CNN 2, the second cell of CNN 1 to the second cell of CNN 2 etc.).

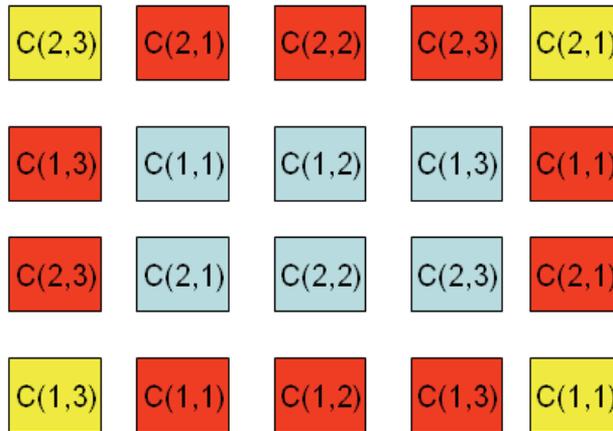


Figure 14. We assume that each CNN has periodical boundary conditions and that cells on the boundaries are connected to other cells in their neighborhood as shown in the Figure.

For example the neighborhood $S_{11}(1)$ for cell C_{11} is given by:

$$S_{11}(1) = \{C_{23}, C_{21}, C_{22}, C_{13}, C_{11}, C_{12}, C_{23}, C_{21}, C_{22}\}$$

5.2 The Genetic Algorithm

We conducted a number of simulations in which we evolved both the first and the second CNN. At the current stage in our work, we have no way of co-evolving the controllers. We obtained the best results when we began by evolving the first CNN and used the results from the best individual to evolve the second CNN.

We used a population of 30 individuals. On every step in the evolutionary process, the population included elite of 10 individuals which did not evolve during that step, 10

individuals generated from the elite by mutation and cross-over and 10 new, randomly generated individuals. Each simulation lasted 20 seconds. The probability of mutation was 3%. The number of cross-over points was set to 3.

5.3 Results

Multiple simulations produced fairly homogeneous results. The GA was reasonably effective in producing a robot with the ability to move rapidly away from its initial position. Evolving of CNN2 for 80 steps, we observed a rapid increase in fitness for the first 15 steps, up to a maximum fitness of approximately 3.5 (and a mean fitness for the elite of 3). Figure 15 shows the fitness achieved by the best individual and the mean fitness for the elite.

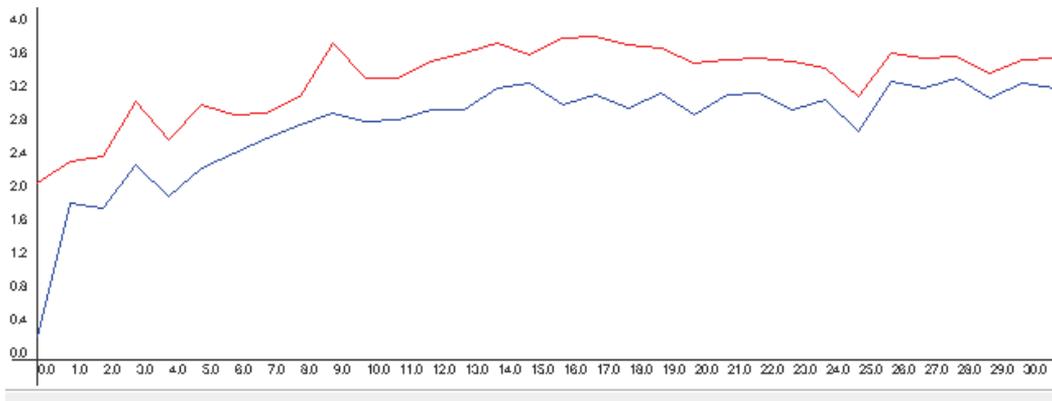


Figure 15. Fitness for the best individual and the mean fitness of the elite

Fig. 15 shows the fitness of the best individual in each generation. As can be seen from the graph, the highest fitness achieved by any individual in any generation was 3.420.

Best individual: G3274, Parent1: G3229, Parent2: G3255

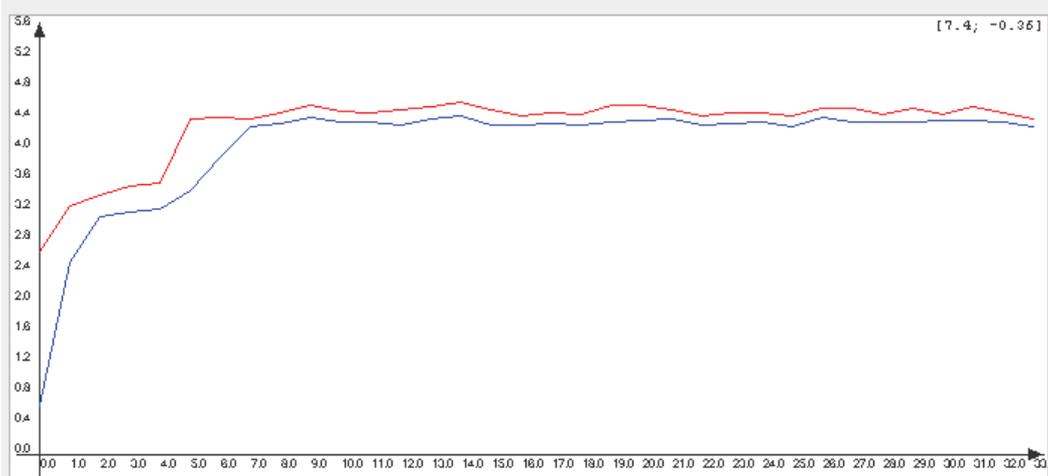


Figure 16. Evolution of CNN1

Following this initial step we took the best individuals from the last generation and continued the evolutionary process for another 30 steps, this time by evolving CNN 1. The

results are shown in Figure 18. After about 7 steps the fittest individual reached a peak of fitness which remained unchanged for the rest of the simulation; after roughly the same number of steps the mean fitness of the elite also reached a ceiling. The maximum fitness achieved was approximately 4.5. The mean fitness of the elite was close to the fitness of the best individual (see Figure 16).

6. On chip prototypes

The CNN Universal Machine (CNN-UM) architecture, introduced by Roska & Chua in 1993 (Roska & Chua, 1993), is an extension of the original CNN paradigm (Chua & Yang., 1988) that has proved extremely effective in many image processing applications. Many different implementations have been proposed. One of the most efficient uses an analog VLSI. With state of the art miniturization it has proved possible to implement a 128*128 or even a 1000*1000 CNN array on a 1 or 2 cm strip of silicon (Rodríguez-Vázquez et al, 1998; Ioan et al., 2001; Paasio et al, 1997). These chips are extremely fast, supporting the equivalent of 1012 billion (1 tera) floating point equivalent operations per second - equivalent to the computational capacity of a supercomputer with 9200 Pentium chips, at the time of writing the fastest in the world (Intel, 2007).

Optical implementations (Andersson, 1998) provide large image resolutions, and make it possible to use larger templates. Feed forward templates compute at the speed of light, providing very fast computation. With feedback templates, by contrast, the optical feedback signal has to be amplified electronically, slowing down the system. Compared to purely electronic systems, these optical systems are relatively bulky and fragile. Systems that use digital hardware emulate CNN-UMs (Wen et al., 1994; Ikenaga & Ogura, 1996; Doan et al., 1994; Adaptive Solutions Inc, 2007; Zarandy et al., 1998) while slower than their analog counterparts, are also more versatile. Given that they use standard digital CMOS technology they are also much quicker to design. One example is the CASTLE architecture (Zarandy et al., 1998) which solves complex image processing problems for medium resolution video streams. Assuming a 25fps digital video feed, CASTLE is fast enough to perform 500 CNN iterations (3x3 convolutions) on frames with 12-bit precision) on each 240x320frame.

7. Conclusions

In this chapter, we have shown how dynamical systems can be used to explore the layering of the sensorial and perceptual activity underlying intelligent behavior in simulated and physical robots. We have described RoVEN, an open, integrated simulation environment which can be easily extended with additional functionality. We have shown some of the results that can be achieved by using Genetic Algorithms to evolve controllers for robot controllers. This represents a first direction in the direction we are seeking to follow. The initial results appear to be satisfactory.

Our implementation work represents an extension of previous work in bio-inspired robotics. As a next step, we intend to build artificial organisms endowed with a cognitive architecture, and a rich sensorial system allowing the agent to recognize and discriminate between specific emotional stimuli from the environment, other agents and humans. These robots will have the ability to generate a set of different motor behaviors, in both simulated and physical environments. The architecture will be multiple-layered, based on modules of interconnected CNN devices. As a result, the robots' cognitive system will be dynamical,

adaptive and self-organizing. The robot sensor system will acquire information from the environment thereby helping to provide the robot with vision, audition, recognition and consciousness. Data from the robot sensory system will be processed in specialized processing units, equivalent to brain areas. It will also be sent to cognitive perceptual centers where it will activate different kinds of behavior. This will make it possible to implement a motor-perception cycle, in which emotional- motivational modules select actions in response to species-specific stimuli from the environment. The system will also include a communication module used to interact with other peers or with humans. The strategic goal is to create new generation of emotional, communicating robots with high-level cognitive capabilities that enable them to achieve complex goals in complex environments, using limited computational resources.

In this scenario, it will be possible to create *artificial brains* with cellular architectures related to a set of basic functionalities which define a biological agent - with evolving or co-evolving modules - which allow for the simulation of the growth and the adaptation of the robot to the environment. These architectures will share a set of common properties such as topographic cellular morphism, different dynamics of communication among cells or layers of the structures and many working non-linear dynamics. Furthermore, it will be possible to combine continuous and discrete robot's representation, integrating algorithmic and physical action in space and time. In this way, the CNN paradigm will make it possible to achieve a vast increase in robot processing power and in the number of cognitive processes that can run in parallel.

8. References

- Adaptive Solutions Inc (2007). CNAPS/PCI Parallel Co-processor
- Andersson, S. (1998). Recent Progress on Logic and Algorithms for Optical Neural Networks (ONN), *Proc. of the fifth IEEE Int. Workshop on Cellular Neural Networks and their Applications*, London.
- Arena, P.; Branciforte, M. & Fortuna, L. (1998). A CNN based experimental frame for patterns and autowaves, *International Journal on Circuit Theory and Applications*, n°3, pp. 120-136.
- Arena, P. & Fortuna, L. (2002). Analog Cellular Locomotion Control of Hexapod Robots, *IEEE Control Systems Magazine*, pp. 21-36
- Bilotta, E.; Cutrì, G. & Pantano, P. (2006). Evolving robot's behavior by using CNNs, *Proc. The Ninth International Conference on the SIMULATION OF ADAPTIVE BEHAVIOR (SAB'06) - FROM ANIMALS TO ANIMATS 9*, 25 - 29 September 2006, CNR, Roma, Italy, *Lecture Notes in Artificial Intelligence*, pp. 631-639
- Bilotta, E.; Pantano, P. & Stranges, F. (2007a). A gallery of Chua's Attractors - Part I, *International Journal of Bifurcation and Chaos*, vol. 17, n° 1, pp. 1-60
- Bilotta, E.; Pantano, P. & Stranges, F. (2007b). A gallery of Chua's Attractors - Part II, *International Journal of Bifurcation and Chaos*, vol. 17, n° 2, pp. 293-380
- Bilotta, E.; Pantano, P. & Stranges, F. (2007c). A gallery of Chua's Attractors - Part III, *International Journal of Bifurcation and Chaos*, vol. 17, n° 3, 657-734
- Bilotta, E.; Di Blasi, G.; Pantano, P. & Stranges, F. (2007d). A gallery of Chua's Attractors - Part IV, *International Journal of Bifurcation and Chaos*, vol. 17, n° 4, pp. 1017-1078
- Bilotta, E.; Di Blasi, G.; Pantano, P. & Stranges, F. (2007e). A gallery of Chua's Attractors - Part V, *International Journal of Bifurcation and Chaos*, vol. 17, n° 5, pp. 1383-1511

- Bilotta, E.; Di Blasi, G.; Pantano, P. & Stranges, F. (2007f). A gallery of Chua's Attractors – Part VI, *International Journal of Bifurcation and Chaos*, vol. 17, n° 1, pp. 1801-1910
- Calabrese R.L., (1995). Oscillation in motor pattern generating networks, *Current Opinion in Neurobiology*, vol. 5, pp. 816, 823
- Cangelosi, A. & Parisi, D. (2002). Computer simulation: A new scientific approach to the study of language evolution. In *Simulating the evolution of language*, Cangelosi, A. & Parisi, D. (Ed.), London: Springer, pp. 3-28
- Chua, L.O. & Yang, L. (1988). Cellular neural networks: Theory and Applications, *IEEE Trans. on Circuits and Systems*, vol. 35, pp. 1257-1290
- Chua, L.O. (1998). *CNN: A Paradigm for Complexity*. World Scientific Publishing Co. Pte. Ltd.
- Chua, L.O. (1995). Special issue on nonlinear waves, patterns and spatio-temporal chaos in dynamic arrays, *IEEE Transactions on Circuits and System*, vol. 42, n°10
- Doan, M.D.; Glesner, M.; Chakrabaty, R.; Heidenreich, M. & Cheung, S. (1994). Realisation of digital Cellular Neural Network for image processing, *Proc. of the IEEE CNNA'94*, Rome, pp. 85-90
- Fried, S. I.; Münch, T. A. & Werblin, F., S. (2005). Directional Selectivity is Formed at Multiple Levels by Laterally Offset Inhibition in the rabbit Retina, *Neuron*, vol. 46, pp. 117-127
- Gacsádi A.; Tiponut V. & Szolgay P. (2006). Image-Based Visual Servo Control of a Robotic Arm by Using Cellular Neural Networks, *Proceedings of the 15th International Workshop on Robotics in Alpe-Adria-Danube Region*, (RAAD 2006), ISBN 9637154 48 5, CD Rom, Balatonfüred, Hungary
- Gacsádi A. & Szolgay P. (2004). A variational method for image denoising, by using cellular neural networks, *Proceedings of the IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA 2004)*, Budapest, Hungary, ISBN 963-311-357-1, pp. 213-218
- Harnad, S. (1990). The Symbol Grounding Problem, *Physica D*, vol. 42, pp. 335-346
- Holland, J. (1993). *Adaptation in natural and artificial systems*, Penguin Books
- Ikenaga, T. & Ogura, T. (1996). Discrete-time Cellular Neural Networks using highly-parallel 2D Cellular Automata CAM2, *Proc. of Int. Symp. on Nonlinear Theory and its Applications*, pp. 221-224.
- Intel (2007) <http://www.intel.com/pressroom/archive/releases/CN0611B.HTM>
- Ioan, D.; Duca, A. & Rebican, M. (2001). Teams of Autonomous Software Agents (TASA) to Solve Inverse ENDE Problems. *The Abstracts of Progress In Electromagnetics Research Symposium (PIERS01)*, Osaka, Japan, iul. pp. 18-22.
- Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. The MIT Press
- Keijzer, F. (2002). Representation in dynamical and embodied cognition, *Cognitive Systems Research*, vol. 3, pp. 275–288
- Krinsky, V.I. (1984). *Self-organization: autowaves and structures far from equilibrium*, Springer Ed, Berlin
- Murray, J.D. (1989). *Mathematical biology*, Springer Ed. , Berlin
- Nolfi, S. & Floreano, D. (2000). *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. Cambridge, MA, MIT Press/Bradford Books.
- Paasio, A.; Dawindzuk, A.; Halonen, K. & Porra, V. (1997). Minimum Size 0.5 Micron CMOS Programmable 48x48 CNN Test Chip, *European Conference on Circuit Theory and Design*, Budapest

- Pearson, G.K. (1993). Common principles of motor control in vertebrates and invertebrates, *Annal Review of Neuroscience*, n° 16, pp. 295-297
- Rodriguez-Vazquez, A.; Dominguez-Castro, R. & Espejo, S. (1998). Challenges in Mixed-Signal IC Design of CNN Chips in Submicron CMOS, *Proc. of the fifth IEEE Int. Workshop on Cellular Neural Networks and their Applications*, April, London.
- Roska, T. & Chua, L.O. (1993). The CNN Universal Machine: an analogic array computer, *IEEE Transactions on Circuits and Systems-II*, vol. 40, pp. 163-173
- Roska, B.; Molnara, A. & Werblin, F., S. (2006) Parallel Processing in Retinal Ganglion Cells: How Integration of Space-Time Patterns of Excitation and Inhibition Form the Spiking Output, *Journal of Neurophysiology*, vol. 95, pp. 3810-3822
- Schilling, M.; Cruse, H. & Arena, P. (2007). Hexapod Walking: an expansion to Walknet dealing with leg amputations and force oscillations, *Biol. Cybern.* vol. 96, pp. 323-340
- van Gelder, T. J. (1995). The distinction between mind and cognition. In Houng Y. H. and Ho J. C.(Ed), *Mind and Cognition*. Taipei, Academia Sinica, pp. 57-82.
- van Gelder, T. J. (1998a). The dynamical hypothesis in cognitive science, *Behavioural and Brain Sciences*, vol. 21, pp. 1-14
- van Gelder, T. J. (1998b). Disentangling dynamics, computation, and cognition, *Behavioural and Brain Sciences*, vol. 21, pp. 40-47.
- Wen, K.A.; Su, J.Y. & Lu, C.Y. (1994). VLSI design of digital Cellular Neural Networks for image processing, *Journal of Visual Communication and Image Representation*, vol.5 , n°2, pp. 1117-126
- Zarandy, A.; Keresztes, P.; Roska, T. & Szolgay, P. (1998). An emulated digital architecture implementing the CNN Universal Machine, *Proc. of the fifth IEEE Int. Workshop on Cellular Neural Networks and their Applications*, London, pp. 249-252

Optimal Design of Mechanisms for Robot Hands

J.A. Cabrera, F. Nadal and A. Simón
*University of Málaga, Department of Mechanical Engineering
Spain*

1. Introduction

Scientific progress has been very important in the last two centuries, from the invention of the steam machine until the electronic age. Robots design has also suffered an important progress in the last decades and many approaches deal with this issue. Our approach carries out an optimal design of planar 1 DoF mechanisms which are used for robot hands or grippers. These kinds of mechanisms are amply used because of their simplicity and they only need one motor to move them, so many robots use this kind of mechanism as grippers. We studied a new technique to carry out an optimal design of a gripper in this work. A searching procedure is developed, which applies genetic algorithms based on an evolutionary approach. The new method has proved to solve synthesis problems of planar mechanisms and has been used for testing a hand robot mechanism, showing that the solutions are accurate and valid for all cases.

Different techniques have been used for mechanism synthesis. In graphical techniques the use of the coupler curve atlas (Hrones & Nelson, 1951) who developed the four-bar mechanisms atlas with almost 10,000 curves is especially remarkable. The solution by (Zhang et al., 1984) focused on five-bar geared linkages. These methods are easy and fast to use but at a low precision rate. The first reference addressing analytical methods was made by (Sandor, 1959), followed by (Erdman, 1981), (Kaufman, 1978) and (Loerch et al., 1975). References about the subject by (Freudenstein, 1954), (Beyer, 1963), (Hartenberg & Denavit, 1964) also exist, solving the synthesis problem using precision points to be reached by the coupler point of the mechanism, but these methods restrict the number of precision points in order to allow the solution of the mathematical system to be closed and show problems caused by wrong sequence of the precision points followed. The great increase in computer power has permitted the recent development of routines that apply numerical methods to the minimization of a goal function. One of the first authors who studied these methods was (Han, 1966), whose work was later improved by (Kramer & Sandor, 1975), (Sohoni & Haug, 1982). They optimized one of the most common goal functions: the error between the points tracked by the coupler and its desired trajectory.

The approach to mechanism synthesis presented in this work deals with evolutionary algorithms based on a differential evolution technique. These kinds of algorithms were first introduced by (Holland, 1973, 1975), whose work is included in Goldberg's book (Goldberg, 1989), and they have been extensively and successfully applied to different optimization problems. These methods define a starting population that is improved by approximations to the goal function making use of natural selection mechanisms and natural genetic laws.

The main advantages of these methods are their simplicity in implementing the algorithms and their low computational cost. In addition, there is no need for extensive knowledge of the searching space, as it is continuous, presents local minimums or shows other mathematical characteristics demanded by traditional searching algorithms. Many researches use this technique for optimum mechanisms synthesis, but they apply a single goal function to carry out the optimization problem. The main difference in our approach is that we use several goal functions and constraints, so the optimization problem is more complex and useful for designing hand robots. Multiobjective techniques are used by (Rao & Kaplan, 1986), (Krishnamurty & Turcic, 1992). (Kunjur & Krishnamurty, 1997) use a multiple criteria optimization approach that obtains Pareto-optimal design solution sets. They apply this method to a mechanism dimensional synthesis with two objective functions and three constraints. (Haulin & Vinet, 2003) develop a multiobjective optimization of hand prosthesis four-bar mechanisms. They use the Matlab optimization toolbox and a goal attainment method for the optimization. All of these need a high power calculus and can fail because they might find the solution in a local minimum ending the search without reaching the true optimal solution.

In this work we have developed an evolutionary approach based on Differential Evolution technique (Storn & Price, 1997). Other authors, like (Cabrera et al., 2002) use this technique for the optimum synthesis of four-bar mechanism. (Shiakolas et al., 2005) also uses Differential Evolution for the optimum synthesis of six-bar linkages, but they apply a single goal function to carry out the optimization problem. We use several goal functions and constrains in our approach, so the optimization problem is more complex and useful in a great variety of problems. We apply our algorithm to hand mechanism synthesis in one-DOF robot, but it is possible to use it in different problems, only changing the goal functions and constraints.

2. Optimization method

Evolutionary algorithms (EAs) are different from more normal optimization and search procedures in four ways:

- Evolutionary algorithms work with a coding of the parameter set, not the parameters themselves.
- Evolutionary algorithms search with a population of points, not with a single point.
- Evolutionary algorithms use evaluations of goal functions, not derivatives or other auxiliary knowledge.
- Evolutionary algorithms use probabilistic transition rules, not deterministic rules.

Altogether, these four differences contribute to an evolutionary algorithm's robustness and turn out to be an advantage over other more commonly used techniques.

Definition.-The optimization problem is given by:

$$\begin{aligned}
 & \min F(X) = (f_1(X), f_2(X), \dots, f_n(X)) \\
 & \text{subject to:} \\
 & \quad g_j(X) \leq 0 \quad j = 1, 2, \dots, m \\
 & \quad \Omega : \{x_i \in [li_i, ls_i] \quad \forall x_i \in X\}
 \end{aligned} \tag{1}$$

Where f_i are the goal functions, i.e. a set of functions where each one expresses a feature or objective to be optimized, and where each individual, X , obtains a value, its fitness. Furthermore $g_j(\cdot)$ are the constraints defining the searching space.

The strategy of evolutionary methods for optimization problems begins with the generation of a starting population. Each individual (chromosome) of the population is a possible solution to the problem and it is formed by parameters (genes) that set the variables of the problem. Genes can be schematized in several ways. In the first approach by (Holland, 1973, 1975) they are binary chains, so each x_i gene is expressed by a binary code of size n . Another way to express the genes, as done in this work, is directly as real values. All genes are grouped in a vector that represents a chromosome, (Storn & Price, 1997) , (Wright, 1990):

$$X = [x_1 \ x_2 \ \dots \ x_k] \quad \forall x \in \mathfrak{X} \quad (2)$$

Next the starting population has to evolve to populations where individuals are a better solution. This task can be reached by natural selection, reproduction, mutation or other genetic operators. In this work, selection and reproduction are carried out sequentially and mutation is used as an independent process. Now, we will define some basic concepts that are very common in multiobjective optimization.

Definition.-Pareto dominance:

A vector, $X^* \in \Omega$, is said to dominate $Y^* \in \Omega$, denoted $X^* \preceq Y^*$, if and only if $f_i(X^*)$ is partially less than $f_i(Y^*)$, i.e.:

$$\forall i \in \{1, \dots, n\} : \{f_i(X^*) \leq f_i(Y^*)\} \wedge \{\exists i \in \{1, \dots, n\} : f_i(X^*) < f_i(Y^*)\}$$

Definition.- Non-dominated or Pareto-optimal solution:

A vector, $X^* \in \Omega$, is said to be non-dominated if and only if there is no vector which dominates X^* , i.e., $\neg \exists Y^* \in \Omega : Y^* \preceq X^*$

Definition.- Pareto-optimal set:

A set, $P \subset \Omega$, is said to be Pareto-optimal if and only if: $\forall X^* \in P : \neg \exists Y^* \in \Omega : Y^* \preceq X^*$

Now we are qualified to explain the evolutionary algorithms that we propose. This algorithm is based on the Differential Evolution algorithm proposed by (Storn & Price, 1997), but we introduce a set of new features:

- The original Differential Evolution algorithm was used in optimization problems with one goal function. We use it with multiobjective problems.
- We use a Pareto-based approach to sort the population and this one is divided into non-dominated and dominated population. The 'best' individuals are chosen to run the Differential Evolution strategy from the non-dominated sub-population.
- We use a genetic operator called mutation, which is not used in the original algorithm. This operator is of great significance in certain problems to prevent stagnation (Lampinen & Zelinka, 2000).
- We use a function to control the number of non-dominated individuals in the population.
- We introduce a procedure for handling the constraints. This procedure is based on the work proposed by (Lampinen, 2002), but applied to multiobjective problems.

Definition.- Selection of a couple for reproduction:

For selection, two individuals are randomly chosen from the population and they form a couple for reproduction. The selection can be based on different probability distributions, such as a uniform distribution or a random selection from a population where the weight of each individual depends on its fitness, so that the best individual has the greatest probability to be chosen. In this paper, an individual randomly selected from the Pareto-optimal set of the population and two individuals randomly selected from the complete population with uniform distribution are chosen for reproduction and they make up a disturbing vector, V . The scheme, (Storn and Price, 1997), known as Differential Evolution yields:

$$\begin{aligned} P &\equiv \{X_i : i \in [1, NP]\} \\ \zeta &\subset P : Y_{r1} \in \zeta \\ V &= Y_{r1} + F \cdot (Y_{r2} - Y_{r3}) \end{aligned} \quad (3)$$

where Y_{r1} is an individual chosen randomly from the Pareto-optimal set ζ of population P , which is obtained as defined previously, Y_{r2} and Y_{r3} are two individuals randomly selected from population P among NP individuals, and F is a real value that controls the disturbance of the Pareto-optimal individual. This disturbing vector V and individual i of the population form the couple for reproduction. This way to obtain parent V , it maintains the philosophy of the original Differential Evolution algorithms, where the best individual of the population and two individuals chosen randomly are used to obtain the disturbing vector V . In some ways Y_{r1} are the 'best' individuals in the actual population, because they are chosen from the Pareto-optimal set.

Definition.- Reproduction:

Next, for reproduction, V is crossed with individual i of the current population to generate individual i of the next population. This operator is named crossover.

In natural reproduction, parents' genes are exchanged to form the genes of their descendant or descendants. As shown in Figure 1, reproduction is approached by a discrete multipoint crossover that can be used to generate X_i^N : parent X_i^G provides its descendant with a set of genes randomly chosen from its entire chromosome and parent V provides the rest. Crossover is carried out with a probability defined as $CP \in [0, 1]$.

Definition.- Selection of new descendants:

The following steps are performed to choose which individual X_i^N or X_i^G passes to the next population:

- If the new X_i^N descendent fulfills more constraint than parent X_i^G , then the new descendent is chosen for the next population, i.e.,
- if $\exists k : g_k(X_i^N) \leq 0 \rightarrow \xi_k(X_i^N) = 1$, then:
- $\forall k \in \{1, \dots, m\} : \sum \xi_k(X_i^N) > \sum \xi_k(X_i^G) \rightarrow X_i^{G+1} = X_i^N$
- If parent X_i^N fulfills more constraints than the new X_i^G descendent, then the parent is chosen for the next population, i.e., $\forall k \in \{1, \dots, m\} : \sum \xi_k(X_i^N) \leq \sum \xi_k(X_i^G) \rightarrow X_i^{G+1} = X_i^G$
- If both individuals X_i^N and X_i^G fulfill all constraints or do not fulfill some of them, then the individual which dominates is chosen for the next population, i.e.:

$$\left. \begin{aligned} & \left\{ \forall k \in \{1, \dots, m\} : g_k(X_i^N) \leq 0 \wedge g_k(X_i^G) \leq 0 \right\} \\ & \left\{ \vee \left\{ \exists k : g_k(X_i^N) > 0 \wedge g_k(X_i^G) > 0 \right\} \right\} \\ & \wedge \left\{ X_i^N \leq X_i^G \right\} \rightarrow X_i^{G+1} = X_i^N \end{aligned} \right\}$$

$$\left. \begin{aligned} & \left\{ \forall k \in \{1, \dots, m\} : g_k(X_i^N) \leq 0 \wedge g_k(X_i^G) \leq 0 \right\} \\ & \left\{ \vee \left\{ \exists k : g_k(X_i^N) > 0 \wedge g_k(X_i^G) > 0 \right\} \right\} \\ & \wedge \left\{ X_i^G \leq X_i^N \right\} \rightarrow X_i^{G+1} = X_i^G \end{aligned} \right\}$$

Therefore the population neither increases nor decreases.

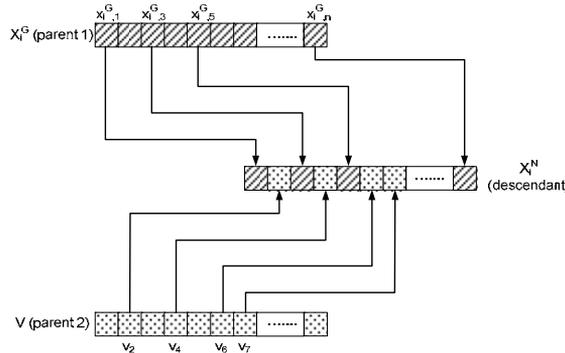


Figure 1. Reproduction scheme based on discrete multipoint crossover

Definition.-Mutation

A new mutation procedure of the parameters to be optimized is developed in this work. Mutation is an operator consisting of random change of a gene during reproduction. We have verified that this procedure is fundamental to obtain the optimum when the parameter range values are very different. The mutation procedure changes only some of these parameters allowing to find the correct optimum and not to stop in a local minimum. This problem was called stagnation in the work performed by (Lampinen & Zelinka, 2000) and it is shown in Figure 2.

The whole procedure to obtain a new descendent is shown in Figure 2a. In this case, there are two different parameters (genes) and the optimum has a very different value for these two parameters. And we suppose that the individuals of the population are situated around a local minimum due to the evolution of the population. The fundamental idea of this discussion consists of the step length adaptability along the evolutionary process. At the beginning of the generations the step length is large, because individuals are far away each from other. As evolution goes on, the population converges and the step length becomes smaller and smaller. For this reason if the mutation procedure does not work properly, it is possible to drop in a local minimum. In Figure 2a and 2b the differences between both strategies with and without mutation procedure are shown.

The way to obtain a new descendent of the next population without mutation procedure is shown in Figure 2a. In this case the V and X_i^G couple generates the X_i^N descendent, but this new chromosome may not reach the global minimum due to the fact that the absolute values of the genes that compose it are very different, and the selection plus reproduction operations are not able to make the new descendent by themselves to overcome the valley of the local minimum.

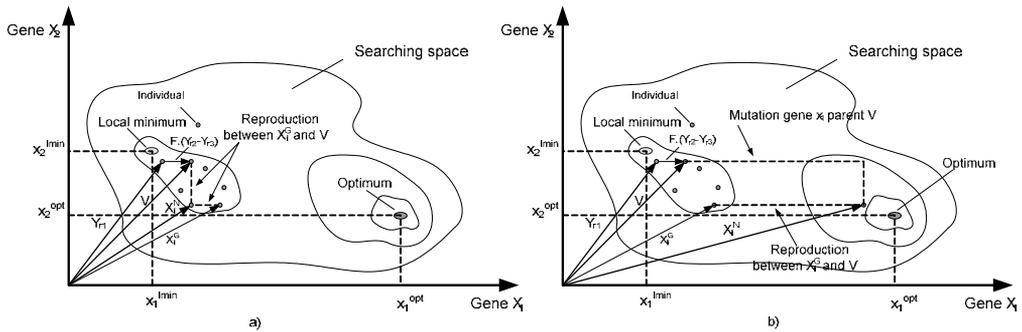


Figure 2. a) Differential Evolution without mutation procedure. b) Differential Evolution with mutation procedure

With the mutation procedure it is possible to solve the problem explained before. The generation of a new descendant using the mutation procedure is schemed in Figure 2b. Here, the value of one or several of the genes of the V and X_i^G couple is changed in a range defined by the user, when the reproduction is taking place. This fact yields a new descendant, X_i^N , which has a different fitness from the X_i^G descendant studied in the previous case. This allows the algorithm to look for individuals with better fitness in the next generation.

In this work, mutation is defined as follows: when gene x_i mutates, the operator randomly chooses a value within the interval of real values $(x_i, x_i \pm \text{range})$, which is added or subtracted from x_i , depending on the direction of the mutation.

Mutation is carried out with a probability defined as $MP \in [0, 1]$, much lower than CP . Once the genetic operators are described, the optimization algorithm will be explained.

3. POEMA algorithm

The proposed algorithm, which is defined as Pareto Optimum Evolutionary Multiobjective Algorithm (POEMA), has the following steps:

1. The algorithm starts with the random generation of a starting population with NP individuals.
2. Next, the algorithm calculates the Pareto-optimal set of the total population and obtains its size, N_{pr} . To preserve diversity, the number of non-dominated individuals is maintained along iterations according to the following function:

$$\forall k \in \{1, \dots, \text{itermax}\}: N_{pr} \leq N_0 + k \cdot \Delta N$$

Where itermax is the number of iterations in the algorithm, N_0 is the number of allowed initial individuals in the Pareto-optimal set and ΔN is a parameter to increase the allowed initial individuals with the iterations. So a maximum number of non-dominated individuals are allowed. If this maximum is exceeded, the nearest neighbor distance function is adopted (Abbass, 2002).

3. To create the new population, the selection of couple, reproduction and mutation operator are used according to definitions described above.
4. If the algorithm reaches the maximum number of iterations, it finishes; otherwise return to step 2.

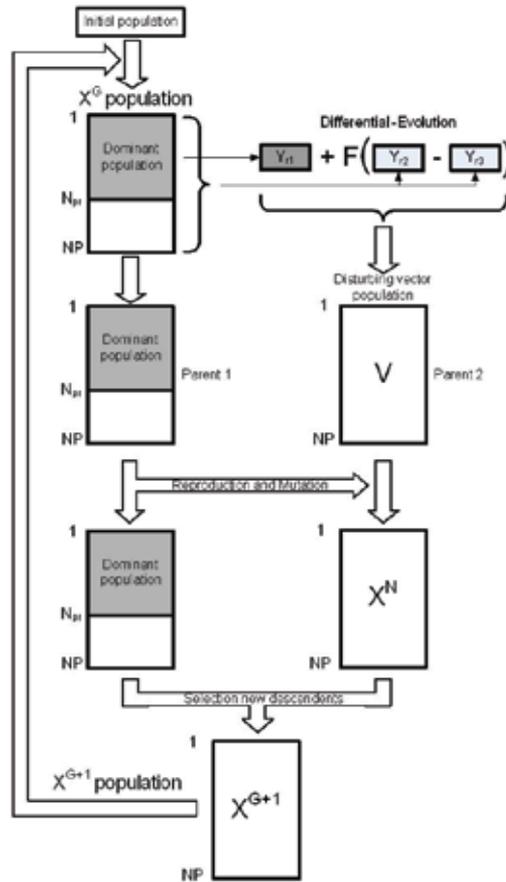


Figure 3. Scheme algorithm

A scheme of the proposed algorithm is shown in figure 3. First, we generate a parent for reproduction according to the Differential-Evolution scheme, which was defined above. Hence the couple for reproduction is the actual population, X^G , and the disturbing vector population is V . As the reproduction and mutation operator are carried out, a new population is obtained, X^N . This one is compared with the actual population, X^G , to obtain the new population, X^{G+1} . At this point, we obtain the Pareto-optimal set of the new population according to what we explained above and we run a new cycle in the algorithm. As we can observe, the new population maintains the same number of individuals as the previous one, so this algorithm does not increase the number of individuals in the population.

4. Goal function and constraint formulation in the two proposed problems

Once we have described the POEMA algorithm, we will develop the goal functions for the problem of a robot hand mechanism in this section. The advantage of using a multiobjective evolutionary algorithm is that we can include either kind of goal function that other works have resolved individually. When a mechanism is designed, several kinds of features are kept in mind:

- Geometric features: a link has to measure a specific length, etc.

- Kinematical features: a point in the mechanism has to follow a specific trajectory, velocity or acceleration law during its movements.
- Mechanical advantage: amount of power that can be transmitted by the mechanism for one complete cycle.

First problem.- In this problem we dealt with a robot hand mechanism (Figure 4). The goal functions for this problem were:

1.- A grasping index (GI) that is similar to the mechanical advantage concept, (Ceccarelli, 1999), which is obtained by means of the method of virtual work applied between the input slider (point F) and the output link (point E), obtaining:

$$-P \cdot v_y^F = 2 \cdot (F \cdot \cos \psi \cdot v_x^E + F \cdot \sin \psi \cdot v_y^E) \Rightarrow GI = \frac{2 \cdot F \cdot \cos \psi}{P} = \frac{v_y^F}{v_x^E + v_y^E \cdot \tan \psi} \quad (4)$$

As we can see in the previous equation, the *GI* grasping index must be maximized. However, we will convert this objective in a minimizing function, so the first goal function is:

$$f_1 = \min \left(\frac{v_x^E + v_y^E \cdot \tan \psi}{v_y^F} \right) \quad (5)$$

2.- The second objective is to minimize the acceleration in the *E* contact point to avoid a big impact on the object. So the second goal function is:

$$f_2 = \min(a_x^E) \quad (6)$$

3.-Another objective is to reduce the weight of the mechanism. If we consider all the links with the same thickness, this objective will be:

$$f_3 = \min \sqrt{\sum_{i=1}^n (x_i^2)} \quad (7)$$

Where x_i is the length of the *i* link in the mechanism.

4.- The last objective is to standardize the link length in the mechanism to avoid a great difference between the length of the different links. So the fourth goal function is:

$$f_4 = \min \frac{\sqrt{\sum_{i=1}^n \sum_{j=1}^n (x_i - x_j)^2}}{\sqrt{\sum_{i=1}^n (x_i^2)}} \quad (8)$$

The constraints for this problem are:

$$g_1 \equiv v_x^E > 0 \quad (9)$$

$$g_2 \equiv EJ = D_{mec} \quad (10)$$

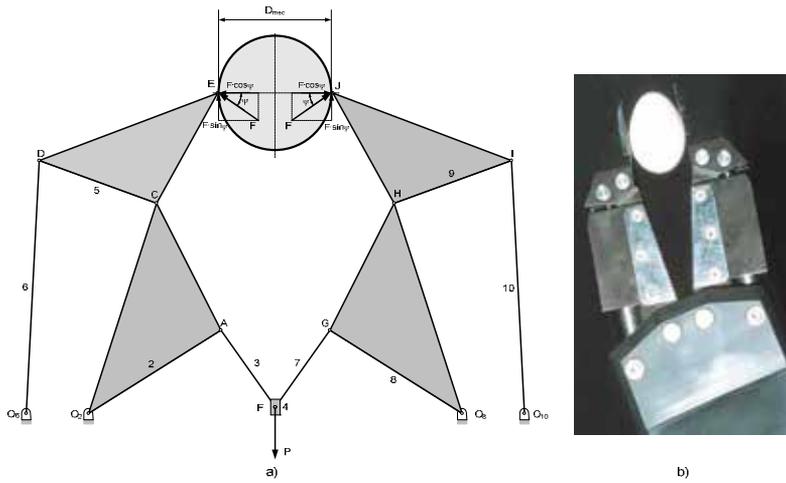


Figure 4. a) Hand robot mechanism for problem 1 and 2. b) Real robot manipulator

The first constraint says that the velocity of the E contact point must go in the same direction as the grasping. And the second constraint says that the distance between the two contact points, \overline{EJ} , must be object size D_{mec} . Hence the whole optimization problem is shown in the next equation:

$$\begin{aligned}
 & \min \{f_1(X), f_2(X), f_3(X), f_4(X)\} \\
 & \text{subject to :} \\
 & g_1(X) \\
 & g_2(X) \\
 & l_{\min} \leq X \leq l_{\max}
 \end{aligned} \tag{11}$$

Where, the X vectors are the design variables. We also introduce a boundary constraint of the design variables. To find the design variables, it is necessary to do a kinematic analysis of the mechanism. We use the Raven method to determine the position, velocity and acceleration of the E contact point, because these variables are in f_1 and f_2 goal functions in the first problem and in f_1 , f_2 and f_3 goal functions in the second problem. The rest of the goal functions in both problems only need the link lengths of the mechanism. Hence, we establish the following scheme according to Figure 5:

$$\vec{E} = \vec{r}_{1x} + \vec{r}_{1y} + \vec{r}_6 + \vec{g}_6 \tag{12}$$

Then to obtain the position of the contact point E :

$$E_x = -r_{1x} + r_6 \cdot \cos \theta_6 + g_6 \cdot \cos(\theta_5 + \delta) \tag{13}$$

$$E_y = r_{1y} + r_6 \cdot \sin \theta_6 + g_6 \cdot \sin(\theta_5 + \delta) \tag{14}$$

$$\psi = f(\theta_5, \delta, g_6, r_5) \tag{15}$$

To obtain the velocity of the contact point E:

$$v_x^E = -r_6 \cdot \omega_6 \cdot \sin \theta_6 - g_6 \cdot \omega_5 \cdot \sin(\theta_5 + \delta) \tag{16}$$

$$v_y^E = r_6 \cdot \omega_6 \cdot \cos \theta_6 + g_6 \cdot \omega_5 \cdot \cos(\theta_5 + \delta) \tag{17}$$

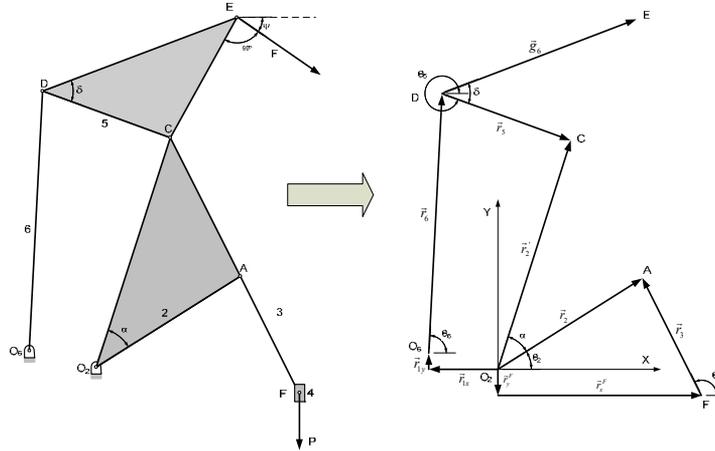


Figure 5. Kinematic study of the mechanism

And the acceleration:

$$a_x^E = -r_6 \cdot \omega_6^2 \cdot \cos \theta_6 - r_6 \cdot \alpha_6 \cdot \sin \theta_6 - g_6 \cdot \omega_5^2 \cdot \cos(\theta_5 + \delta) - g_6 \cdot \alpha_5 \cdot \sin(\theta_5 + \delta) \tag{18}$$

In the previous equations we have obtained all the variables that we need in the two problems proposed, but we also have to develop the following schemes:

$$\vec{r}_y^F + \vec{r}_x^F + \vec{r}_2 = \vec{r}_3 \tag{19}$$

$$\vec{r}_{1x} + \vec{r}_{1y} + \vec{r}_6 + \vec{r}_5 = \vec{r}_2' \tag{20}$$

The first equation solves the slider-crank mechanism (2-3-4) and the second one solves the four-link mechanism (2-5-6), so the design variables for the proposed problems are:

$$X = \{r_x^F, r_y^F, r_2, r_3, r_{1x}, r_{1y}, r_2', r_5, r_6, \alpha, \delta, g_6\} \tag{21}$$

The design variables are the same for the two problems. The only difference is that in the second problem the variable r_x^F , what it is the actuator position, has different positions to obtain different contact point positions.

Second problem.- In this problem we will use the same hand robot mechanism (Figure 4), but in this case the mechanism will be able to grasp different objects with different sizes, i.e., the size of the object is within a determined range. Hence, in this problem the input slider has different positions that determine the different positions (precision points) of the output link. In this case the goal functions are:

1.- As the mechanism is in movement and the output link has different positions, i.e., the E contact point follows several precision points to determine the range of the size of the object,

we will try to make the E contact point follow a determined trajectory as well, so the first goal function is:

$$f_1 = \min \left(\sum_{i=1}^n \sqrt{(E_{x,i}^{obj} - E_{x,i}^{mech})^2 + (E_{y,i}^{obj} - E_{y,i}^{mech})^2} \right) \quad (22)$$

Where $E_{x,i}^{obj}$ and $E_{y,i}^{obj}$ are the x and y coordinates that the E contact point has to follow in each i position and $E_{x,i}^{mech}$ and $E_{y,i}^{mech}$ are the x and y coordinates of the E contact point of the designed mechanism in each i position. Hence, this goal function measures the error between the desired trajectory and the mechanism trajectory of the E point.

2.-The second goal function minimizes the grasping index (GI) developed in the previous problem, but applied to each i position of the E contact point, i.e, we obtain an average grasping index.

$$f_2 = \min \left(\frac{\sum_{i=1}^n \frac{v_{x,i}^E + v_{y,i}^E \cdot \tan \psi_i}{v_{y,i}^E}}{n} \right) \quad (23)$$

In this case we obtain v_x^E , v_y^E , v_y^F and ψ_i in each i precision point. And n is the number of precision points. The following goal functions are the same as in the previous problem:

$$f_3 = \min \left(\sum_{i=1}^n a_{x,i}^E \right) \quad (24)$$

$$f_4 = \min \sqrt{\sum_{i=1}^n (x_i^2)} \quad (25)$$

$$f_5 = \min \frac{\sqrt{\sum_{i=1}^n \sum_{j=1}^n (x_i - x_j)^2}}{\sqrt{\sum_{i=1}^n (x_i^2)}} \quad (26)$$

In this problem the constraints are related to the velocity of the E contact point. This velocity, as in the previous problem, must be greater than zero, but in this case we have different E contact point velocities, so we have as many constraints as precision points.

$$\begin{aligned} g_1 &\equiv v_{x,1}^E > 0 \\ g_2 &\equiv v_{x,2}^E > 0 \\ &\vdots \\ g_n &\equiv v_{x,n}^E > 0 \end{aligned} \quad (27)$$

Once the problem has been defined, we can show it in the next equation:

$$\begin{aligned}
 & \min \{f_1(X), f_2(X), f_3(X), f_4(X), f_4(X)\} \\
 & \text{subject to:} \\
 & g_1(X) \\
 & g_2(X) \\
 & \vdots \\
 & g_n(X) \\
 & l_{\min} \leq X \leq l_{\max}
 \end{aligned} \tag{28}$$

As in the previous problem, the X vectors are the design variables.

5. Results

In the first place, we show the results of the first problem. In this case, the algorithm parameters are: (number of individuals in the population) $NP=100$, (maximum iteration number) $itermax=5000$, (disturbing factor) $F=0.5$, (crossover probability) $CP=0.2$, (mutation probability) $MP=0$, (initial number of non-dominated individuals) $N_o=40$, (non-dominated individual growth) $\Delta N=0.012$, (size of the object) $D_{mec}=100$, (actuator velocity) $v_y^F=-1$, (actuator acceleration) $a_y^F=1$.

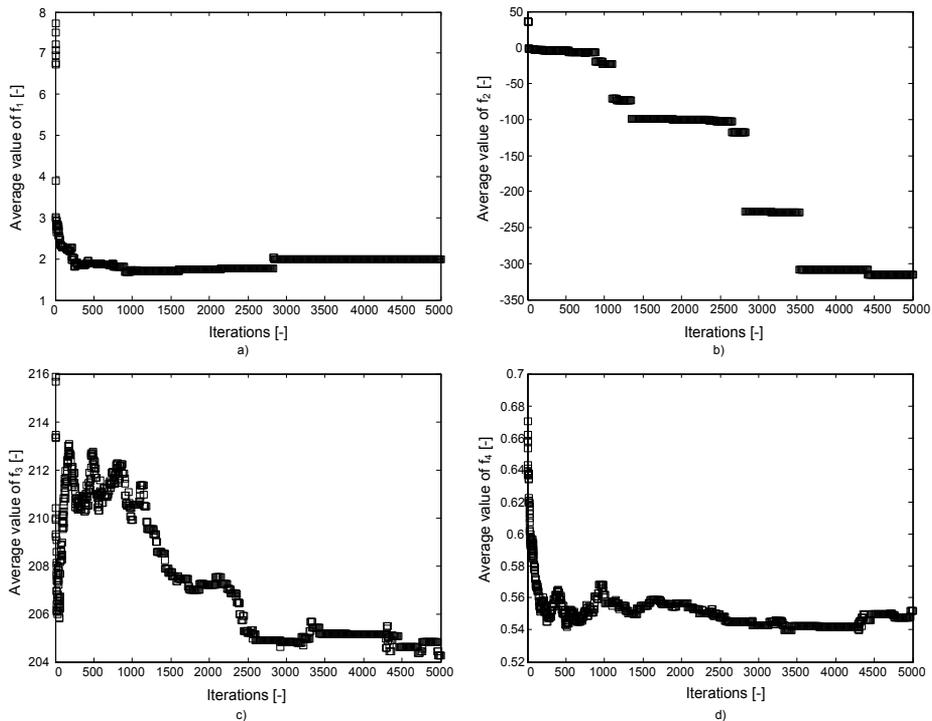


Figure 6. Average value evolution of the goal functions along iterations in the first problem

We show the average value evolution of the goal functions along iterations in Figure 6. The average values are obtained by the following equation:

$$f_{k,iter} = \frac{\sum_{i=1}^{NP} f_k(X_i^{iter})}{NP} \quad (29)$$

Where $k \in \{1,2,3,4\}$ are the numbers of goal functions in the first problem and NP is the number of individuals in the population.

We show the average value behavior of the goal functions in the previous figure. We can observe how the average values have decreased in every case. The average values are mean values of the goal functions of the all the individuals in the population and at the beginning of the iterations there are a few 'good individuals', i.e., non-dominated individuals, so those values are bad when the algorithm starts the iterations and they improve when the algorithm finishes. However, the algorithm does not always finish with the best average values as we can see in Figure 6a and 6d. This fact happens because the number of non-dominated individuals is not the total number of individuals in the population and the average values can be worse in the final populations because the dominated individuals in the final populations make the average value worse.

The non-dominated individuals' behavior can be observed in Figure 7. We can see how non-dominated individuals at the beginning of the iterations follow the established law, increasing the number of non-dominated individuals linearly with the iterations. At the end of the iterations, the number of non-dominated individuals is lower than the allowed non-dominated individuals. Hence the non-dominated individuals in the final populations are not the whole number of individuals in the population.

We also show the three 'best' mechanisms of the final population in the following figure. We draw the mechanisms that have the best value of one of the goal function values, but this does not mean that these mechanisms are the best mechanisms in the final population, as the final population has about eighty-four non-dominated mechanisms (see Figure 7).

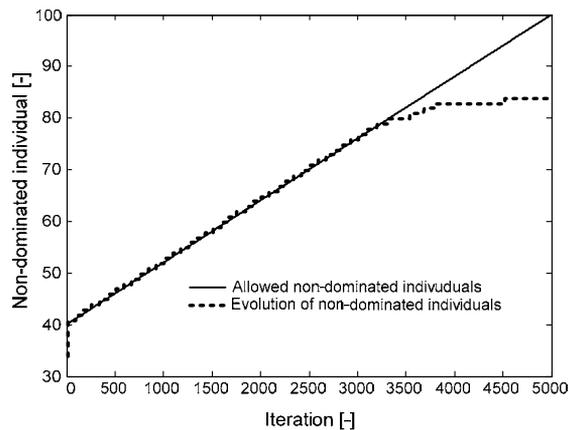


Figure 7. Evolution of non-dominated individuals along iterations

The design variable values of these three mechanisms are shown in the following table:

	r_x^F [L]	r_y^F [L]	r_2 [L]	r_3 [L]	r_{1x} [L]	r_{1y} [L]	r_2' [L]	r_5 [L]	r_6 [L]	α [-]	δ [-]	g_6 [L]
(a)	47.9	-15.3	34.2	42.7	-69.1	-17.4	81.8	63.4	124.6	1.3	1.38	134.4
(b)	50.2	-1.42	32.1	37.6	-41.4	-2.77	87.7	39.2	109.9	0.8	1.23	54.41
(c)	49.7	-10.9	33.9	35.1	-68.6	-20.6	80.5	55.7	93.21	1.2	0.84	117.8

Table 1. Design variable values of the three selected mechanisms in the first problem

And the values of every goal function for the three drawn mechanisms are shown in Table 2. The design variables and the goal function values of the three mechanisms correspond to the three mechanisms drawn in Figure 8. As we can see, mechanism (b) has the minimum value of the f_1 and f_2 functions, i.e., it has the minimum value of the contact point acceleration and the minimum value of its dimensions, but it has the worst value of grasping index f_1 and of link proportion f_4 . Instead, mechanism (a) has the best grasping index and mechanism (c) has the best link proportion. Also, the contact point distances are shown in Figure 8. These distances are similar to the three cases and they are very close to our objective.

	f_1 [-]	f_2 [L/T ²]	f_3 [L]	f_4 [-]
Mechanism (a)	0.4207	-0.1266	234.2706	0.4183
Mechanism (b)	1.7828	-901.79	175.9284	0.7677
Mechanism (c)	1.0479	-617.61	205.5393	0.3581

Table 2. Goal function values from three selected mechanisms in the first problem

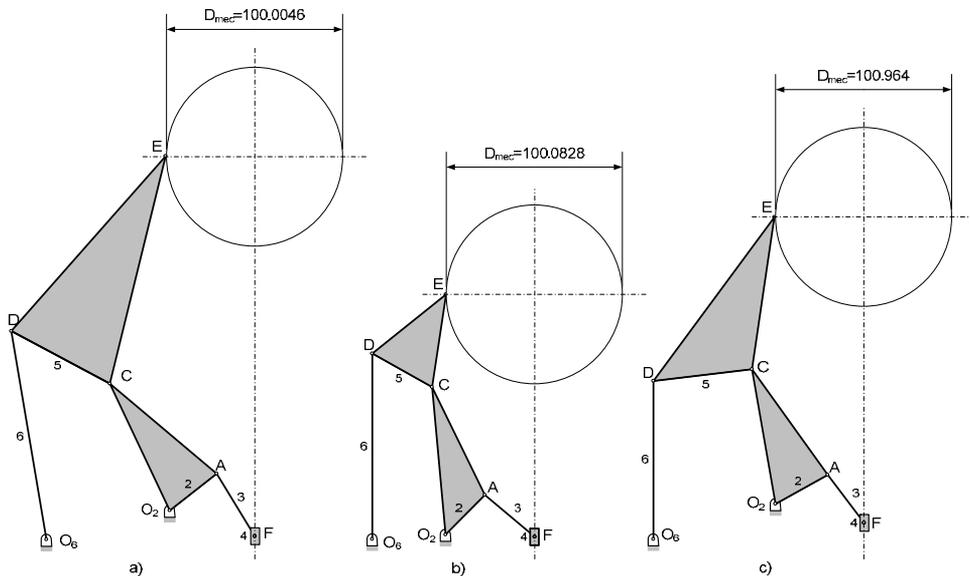


Figure 8. Three mechanisms of the final population in the first problem

We have to highlight that the three selected mechanisms are the ones with the best values of one of the goal function values, but this fact does not imply that these mechanisms are the best among the eighty-four non-dominated mechanisms. Hence, the designer will have to choose which mechanism among the non-dominated mechanisms is the best for him.

Now, we show the results of the second problem. In this case, the algorithm parameters are: (number of individuals in the population) $NP=100$, (maximum iteration number) $itermax=5000$, (disturbing factor) $F=0.5$, (crossover probability) $CP=0.2$, (mutation probability) $MP=0$, (initial number of non-dominated individuals) $N_o=40$, (non-dominated individual growth) $\Delta N=0.012$, (actuator velocity) $v_y^F=-1$, (actuator acceleration) $a_y^F=1$. Again, we show the average value evolution of the goal functions along iterations in Figure 9. The average values are obtained the same way as in the previous problem.

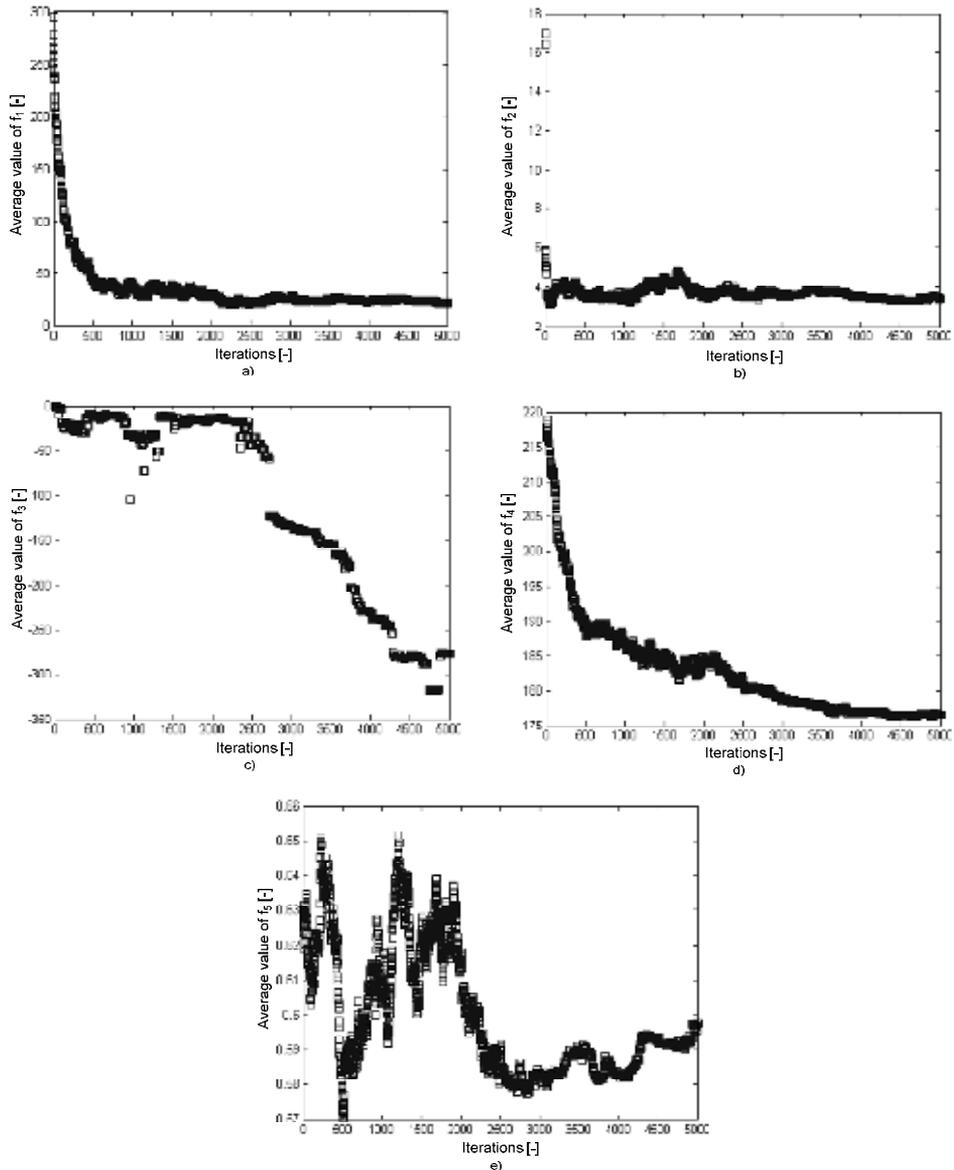


Figure 9. Average value evolution of the goal functions along iterations in the second problem

In this case, the average values of the goal functions also decrease along with the iterations in every case. Only the f_3 and f_5 goal functions have a different behavior when the algorithm finishes and the average values are worse than in the previous iterations. Instead, the new f_1 goal function has a clear decreasing behavior and the average value in the final iterations is the best one.

At the end we show three mechanisms of the final non-dominated population which have the best value of one of the goal functions (Figure 10). In this case, the mechanisms have to follow certain precision points:

$$E_x^{Obj} = \{-10, 10, 40\} \quad E_y^{Obj} = \{160, 170, 165\}$$

The coordinates of the previous precision points are measured from the O_6 fixed point in the mechanism. Hence the design variable values are:

	r_x^F [L]	r_y^F [L]	r_2 [L]	r_3 [L]	r_{1x} [L]	r_{1y} [L]	r_2' [L]	r_5 [L]	r_6 [L]	α [-]	δ [-]	g_6 [L]
(a)	22.5	-6.96 3.04 13.04	28.5	27.0	-57.6	-25.4	76.1	54.27	89.6	0.6	1.18	83.3
(b)	26.03	-12.6 -2.59 7.41	19.36	21.73	-45.2	-36.3	72.7	43.62	92.3	0.9	0.99	78.7
(c)	32.74	-19.3 -9.29 0.71	32.56	20.21	-20.9	-0.57	92.8	40.15	125.0	1.4	1.95	50.8

Table 3. Design variable values of three selected mechanisms in the second problem

The r_y^F value has three positions in the previous table because the E contact point is compared in these three positions of the input slider.

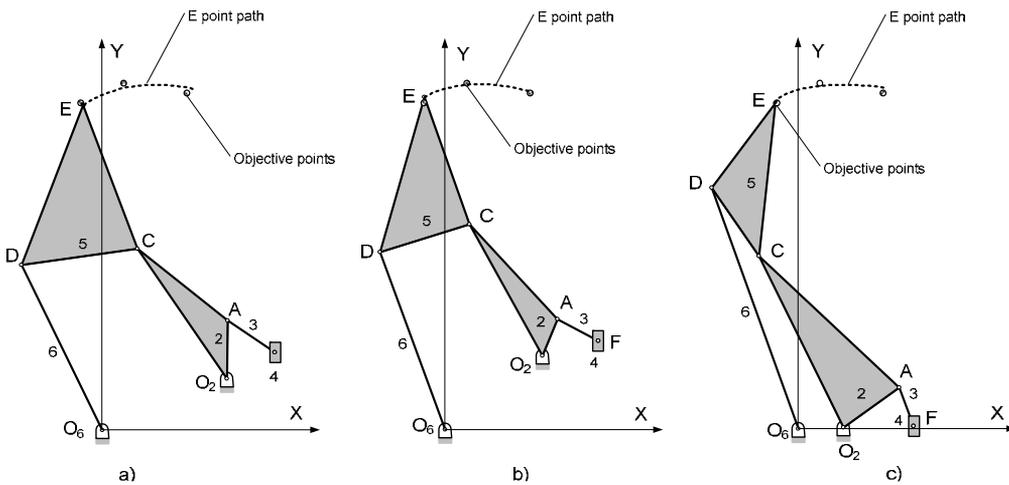


Figure 10. Three mechanisms of the final population in the second problem

We also show the goals function values of these three mechanisms.

Mechanism (a) has the best value of the f_3 and f_5 goal functions, i.e., this mechanism has the minimum value of E contact point acceleration and it has the best link proportion. Instead, mechanism (b) has the best f_1 and f_4 goal functions, so the E contact point path fits objective points more accurately and it also has the minimum value in its dimensions. Finally, mechanism (c) has best average f_2 grasping index.

	f_1 [L]	f_2 [-]	f_3 [L/T ²]	f_4 [L]	f_5 [-]
Mechanism (a)	3.86	1.33	-0.4051	172.38	0.4125
Mechanism (b)	3.54	1.97	-0.1413	163.67	0.5620
Mechanism (c)	3.66	0.1412	-0.0638	177.21	0.9785

Table 4. Goal function values of three selected mechanisms in the second problem

6. Conclusions

In this paper, we showed a new algorithm (POEMA) based on the Differential Evolution strategy, but it has been extended to tackle multiobjective optimization problems. For this purpose, new features have been developed. This work uses the Pareto-based approach to classify the population into non-dominated and dominated individuals.

The algorithm is used to optimize several goal functions in a hand robot mechanism, subject to different constraints. The same method can be applied to optimize any other goal functions in other different problems.

One of the features of the used method is that there is not an unique solution to the problem, as the method finds several solutions which are called non-dominated solutions and every non-dominated solution is a good solution to the proposed problem. Hence, the designer must choose which is the best in every case, i.e., he must determine which characteristic or goal function is a priority and which is not.

An individual evolution study has been made and the obtained results have been satisfactory. We have shown several final mechanisms to the two proposed problems and each one has a good value of one or more features or goal functions.

Another advantage depicted by the method is its simplicity of implementation and that it is possible to use the method in other different mechanism problems by simply changing the goal function formulation for those problems.

7. References

- Hrones, J.A. and Nelson, G.L. (1951). Analysis of the Four bar Linkage, MIT Press and Wiley, New York.
- Zhang, C., Norton R.L. and Hammond, T. (1984). Optimization of Parameters for Specified Path Generation Using an Atlas of Coupler Curves of Geared Five-Bar Linkages, *Mechanism and Machine Theory* 19 (6) 459-466.
- Sandor, G.N. (1959). A General Complex Number Method for Plane Kinematic Synthesis with Applications, *PhD Thesis*, Columbia University, New York.
- Erdman, A.G. (1981). Three and Four Precision Point Kinematic Synthesis of Planar Linkages, *Mechanism and Machine Theory* 16 (5) 227-245.

- Kaufman, R.E. (1978). Mechanism Design by Computer, *Machine Design Magazine*. 94-100.
- Loerch, R.J., Erdman, A.G., Sandor, N., Mihda, A. (1975). Synthesis of Four Bar Linkages with Specified Ground Pivots, *Proceedings of 4th Applied Mechanisms Conference*, Chicago. 101-106.
- Freudenstein, F. (1954). An Analytical Approach to the Design of Four-Link Mechanisms, *Transactions of the ASME* 76. 483-492.
- Beyer, R. (1963). *The Kinematic Synthesis of Mechanism*, McGraw-Hill, New York.
- Hartenberg, R. and Denavit, J. (1964). *Kinematic Synthesis of Linkages*, McGraw-Hill, New York.
- Han, C. (1996). A General Method for the Optimum Design of Mechanisms, *Journal of Mechanisms*. 301-313.
- Kramer, S.N. and Sandor, G.N. (1975). Selective Precision Synthesis. A General Method of Optimization for Planar Mechanisms, *Journal of Engineering for Industry* 2. 678-701.
- Sohoni, V.N. and E.J. Haug, E.J. (1982). A State Space Technique for Optimal Design of Mechanisms, *ASME Journal of Mechanical Design* 104. 792-798.
- Holland, J.H. (1973) Genetic Algorithms and the Optimal Allocations of Trials, *SIAM Journal of Computing* 2 (2) . 88-105.
- Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Michigan.
- Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, Massachusetts.
- Rao, S.S. and Kaplan, R.L. (1986). Transaction ASME Journal of Mechanisms Transmission Automatic Des. 454-460.
- Krishnamurty, S. and Turcic, D.A. (1992). Optimal Synthesis of Mechanisms Using Nonlinear Goal Programming Techniques, *Mechanisms and Machine Theory* 27 (5). 599-612.
- Kunjur, A. and Krishnamurthy, S. (1997). A Robust Multi-Criteria Optimization Approach. *Mechanism and Machine Theory*, Vol. 32 No. 7 . 797-810.
- Haulin, E.N. and Vinet, R. (2003). Multiobjective Optimization of Hand Prosthesis Mechanisms. *Mechanism and Machine Theory*, Vol. 38. 3-26.
- Storn, R. and Price, K. (1997). Differential Evolution. A Simple and Efficient Heuristic Scheme for Global Optimization over Continuous Spaces, *Journal of Global Optimization* 11. 341-359
- Cabrera, J.A. Simon, A. and Prado, M. (2002). Optimal Synthesis of Mechanisms with Genetic Algorithms. *Mechanism and Machine Theory* 37. 1165-1177.
- Shiakolas, P.S., Koladiya, D. and Kebrle, J. (2005). On the Optimum Synthesis of Six-bar Linkages Using Differential Evolution and the Geometric Centroid of Precision Positions Technique. *Mechanism and Machine Theory* 40. 319-335.
- Wright, A.H. (1990). Genetic Algorithms for Real Parameter Optimization, *Proceedings of First workshop on the Foundations of Genetic Algorithms and Classifier Systems*, Indiana. 205-218.
- Lampinen, J. and Zelinka, I. (2000). On Stagnation of the Differential Evolution Algorithm. *Proceedings of MENDEL 2000*, pp. 76-83. Brno, Czech.
- Lampinen, J. (2002). A Constraint Handling Approach for the Differential Evolution Algorithm. *Proceedings of the 2002 Congress on Evolutionary Computation*, Vol. 2, pp. 12-17.
- Abbass, H.A. (2002). The Self-Adaptive Pareto Differential Evolution Algorithm. *Proceedings of the 2002 Congress on Evolutionary Computation*, Vol. 1, pp. 831-836.
- M. Ceccarelli, M. (1999). Grippers as Mechatronics Devices. *Advanced in Multibody Systems and Mechatronics*. 115-130.

Evolving Humanoids: Using Artificial Evolution as an Aid in the Design of Humanoid Robots

Malachy Eaton

*Department of Computer Science and Information Systems, University Of Limerick
Ireland*

1. Introduction

This chapter presents a survey of the state of the art in evolutionary humanoid robotics, focusing mainly, but not exclusively on the use of evolutionary techniques for the design of a variety of interesting behaviours in humanoid robots; both simulated and embodied.

It will discuss briefly the increasing importance of setting robot standards and of benchmarking mobile and service robot performances, especially in the case of future humanoid robots, which will be expected to operate safely in environments of increasing complexity and unpredictability.

We will then describe a series of experiments conducted by the author and his colleagues in the University of Limerick involving the evolution of bipedal locomotion for both a simulated QRIO-like robot, and for the Robotis Bioloid humanoid robot. The latter experiments were conducted using a simulated version of this robot using an accurate physics simulator, and work is ongoing in the transfer of the evolved behaviours to the real robot. Experiments have been conducted using a variety of different environmental conditions, including reduced friction and altered gravity.

The chapter will conclude with a look at what the future may hold for the development of this new and potentially critically important research area.

2. Evolutionary humanoid robotics

Evolutionary humanoid robotics is a branch of evolutionary robotics dealing with the application of evolutionary principles to the design of humanoid robots (Eaton M, 2007). Evolutionary techniques have been applied to the design of both robot body and 'brain' for a variety of different wheeled and legged robots, e.g. (Sims, 1994; Floreano and Urzelai, 2000; Harvey, 2001; Pollack et. al ,2001; Full, 2001; Zykov et al.,2004; Lipson et al.,2006; Bongard et al.,2006). For a good introduction to the general field of evolutionary robotics see the book by Nolfi and Floreano (Nolfi and Floreano, 2000).

In this chapter we are primarily concerned with the application of evolutionary techniques to autonomous robots whose morphology and/or control/sensory apparatus is broadly human-like. A brief introduction to the current state of the art with regard to humanoid robotics including the HRP-3, KHR-1 and KHR-2, Sony QRIO and Honda ASIMO and P2 is contained in (Akachi et al ,2005). Also see the articles by Brooks (Brooks et al.,1998; Brooks,2002) and Xie (Xie et.al,2004) for useful introductory articles to this field.

There are several possible motivations for the creation of humanoid robots. If the robot has a human-like form people may find it more easy and natural to deal with than dealing with a purely mechanical structure. However as the robot becomes more human-like, after a certain point it is postulated that small further increases in similarity result in an unnerving effect (the so called "uncanny valley" introduced by Mori (Mori, 1970) and elaborated by MacDorman (MacDorman, 2005)). The effect is seen to be more pronounced in moving robots than in stationary ones, and is thought to be correlated to an innate human fear of mortality. Another reason, suggested by Brooks (Brooks, 1997) and elaborated recently by Pfeifer and Bongard (Pfeifer and Bongard, 2007) is that the morphology of human bodies may well be critical to the way we think and use our intellect, so if we wish to build robots with human-like intelligence the shape of the robot must also be human-like.

Ambrose argues that another reason for building robots of broadly humanoid form is that the evolved dimensions of the human form may be (semi-) optimal for the dexterous manipulation of objects and for other complex motions; he however argues that the human form should not be blindly copied without functional motivation (Ambrose and Ambrose, 2004).

A final, and very practical reason for the creation of future humanoid robots is that they will be able to operate in precisely the environments that humans operate in today, This will allow them to function in a whole range of situations in which a non-humanoid robot would be quite powerless, with all of the inherent advantages that this entails. Brooks discusses this issue further in his later paper on the subject (Brooks et. al, 2004).

The hope is that by using artificial evolution robots may be evolved which are stable and robust, and which would be difficult to design by conventional techniques alone. However we should bear in mind the caveat put forward by Mataric and Cliff (Mataric and Cliff, 1996) that it is important that the effort expended in designing and configuring the evolutionary algorithm should be considerably less than that required to do a manual design for the exercise to be worthwhile.

We now review briefly some of the work already done, and ongoing, in the emerging field of evolutionary humanoid robotics; this list is not exhaustive, however it gives a picture of the current state of the art.

In an early piece of work in this area Bongard and Paul used a genetic algorithm to evolve the weights for a recurrent neural network with 60 weights in order to produce bipedal locomotion in a simulated 6-DOF lower-body humanoid using a physics-based simulation package produced by MathEngine PLC. (Bongard and Paul, 2001) Inputs to the neural network were two touch sensors in the feet and six proprioceptive sensors associated with each of the six joints. Interestingly, in part of this work they also used the genome to encode three extra morphological parameters - the radii of the lower and of the upper legs, and of the waist, however they conclude that the arbitrary inclusion of morphological parameters is not always beneficial.

Reil and Husbands evolved bipedal locomotion in a 6-DOF simulated lower-body humanoid model also using the MathEngine simulator. They used a genetic algorithm to evolve the weights, time constants and biases for recurrent neural networks. This is one of the earliest works to evolve biped locomotion in a three-dimensional physically simulated robot without external or proprioceptive input. (Reil and Husbands, 2002)

Sellers et al. (Sellers et. al. 2003; Sellers et al. 2004) have used evolutionary algorithms to investigate the mechanical requirements for efficient bipedal locomotion. Their work uses a simulated 6-DOF model of the lower body implemented using the Dynamechs library. The

evolutionary algorithm is used to generate the values used in a finite-state control system for the simulated robot. They suggest the use of an incremental evolutionary approach as a basis for more complex models. They have also used their simulations to predict the locomotion of early human ancestors.

Miyashita et al. (Miyashita et al., 2003) used genetic programming (GP) to evolve the parameter values for eight neural oscillators working as a Central Pattern Generator (CPG) interacting with the body dynamics of a 12 segment humanoid model, in order to evolve biped walking. This work is in simulation only, and a maximum of ten steps of walking were evolved, because of the instability of the limit cycles generated.

Ishiguro et al (Ishiguro et al., 2003) used a two stage evolutionary approach to generate the structure of a CPG circuit for bipedal locomotion on both flat and inclined terrain. They used MathEngine to simulate the lower body of a 7-DOF humanoid robot. An interesting aspect of this work is the adaptability of the evolved controllers to gradient changes not previously experienced in the evolutionary process.

Zhang and Vadakkepat have used an evolutionary algorithm in the generation of walking gaits and allowing a 12-DOF biped robot to climb stairs. The algorithm does, however, contain a degree of domain-specific information. The robot is simulated using the Yobotics simulator and the authors claim the performance has been validated by implementation on the RoboSapien robot (Zhang and Vadakkepat, 2003)

Unusually for the experiments described here, Wolff and Nordin have applied evolutionary algorithms to evolve locomotion directly on a real humanoid robot. Their approach is that starting from a population of 30 manually seeded individuals evolution is allowed to proceed on the real robot. Four individuals were randomly selected then per generation and evaluated, the two with higher fitness then reproduce and replace the individuals with lower fitness. Nine generations were evaluated, with breaks between each generation in order to let the actuators rest. The physical robot used was the ELVINA humanoid with 14-DOF; 12 of these were subject to evolution. While the evolutionary strategy produced an improvement on the hand-developed gaits the authors note the difficulties involved in embodied evolution, with frequent maintenance of the robot required, and the regular replacement of motor servos. (Wolff and Nordin, 2002). Following this the authors moved to evolution in simulation using Linear GP to evolve walking on a model of the ELVINA robot created using the Open Dynamics Engine (ODE) (Wolff and Nordin, 2003).

Endo et al. used a genetic algorithm to evolve walking patterns for up to ten joints of a simulated humanoid robot. Work was also done on the co-evolution of aspects of the morphology of the robot. The walking patterns evolved were applied to the humanoid robot PINO. While stable walking gaits evolved these walking patterns did not generally resemble those used by humans (Endo et al., 2002; Endo et al. 2003). They note that an interesting subject for study would be to investigate what constraints would give rise to human-like walking patterns. A characteristic of our own current work is the human-like quality of the walks generated, as commented on by several observers.

Boeing et al. (Boeing et al., 2004) used a genetic algorithm to evolve bipedal locomotion in a 10-DOF robot simulated using the Dynamechs library. They used an approach that has some parallels with our work; once a walk evolved this was transferred to the 10-DOF humanoid robot 'Andy'. However few of the transferred walks resulted in satisfactory forward motion illustrating the difficulties inherent in crossing the 'reality gap'.

Finally, Hitoshi Iba and his colleagues at the University of Tokyo have conducted some interesting experiments in motion generation experiments using Interactive Evolutionary Computation (IEC) to generate initial populations of robots, and then using a conventional GA to optimise and stabilise the final motions. They applied this technique to optimising sitting motions and kicking motions. IEC was also applied to a dance motions which were implemented on the HOAP-1 robot; the kicking motions were confirmed using the OpenHRP dynamics simulator (Yanese and Iba, 2006). They have also demonstrated the evolution of handstand and limbo dance behavioural tasks (Ayedemir and Iba, 2006). The next section introduces our own work in evolving bipedal locomotion and other behaviours in a high degree-of-freedom humanoid robot.

3. Evolving different behaviours in simulation in a high-DOF humanoid

Bipedal locomotion is a difficult task, which, in the past, was thought to separate us from the higher primates. In the experiments outlined here we use a genetic algorithm to choose the joint values for a simulated humanoid robot with a total of 20 degrees of freedom (elbows, ankles, knees, etc.) for specific time intervals (keyframes) together with maximum joint ranges in order to evolve bipedal locomotion. An existing interpolation function fills in the values between keyframes; once a cycle of 4 keyframes is completed it repeats until the end of the run, or until the robot falls over. The humanoid robot is simulated using the Webots mobile robot simulation package and is broadly modelled on the Sony QRIO humanoid robot (Michel, 2004; Mojon, 2003). (See (Craighead et. al., 2007) for a survey of currently available commercial and open-source robot simulators). In order to get the robot to walk a simple function based on the product of the length of time the robot remains standing by the total distance travelled by the robot was devised. This was later modified to reward walking in a forward (rather than backward) direction and to promote walking in a more upright position, by taking the robots final height into account. The genome uses 4 bits to determine the position of the 20 motors for each of 4 keyframes; 80 strings are used per generation. 8 bits define the fraction of the maximum movement range allowed. The maximum range allowed for a particular genome is the value specified in the field corresponding to each motor divided by the number of bits set in this 8 bit field, plus 1. The genetic algorithm uses roulette wheel selection with elitism; the top string being guaranteed safe passage to the next generation, together with standard crossover and mutation. Two-point crossover is applied with a probability of 0.5 and the probability of a bit being mutated is 0.04. These values were arrived at after some experimentation.

Good walks in the forward direction generally developed by around generation 120. The evolved robots have developed different varieties of walking behaviours (limping, side-stepping, arms swinging, walking with straight/flexed knees etc.) and many observers commented of the lifelike nature of some of the walks developed. We are also exploring the evolution of humanoid robots that can cope with different environmental conditions and different physical constraints. These include reduced ground friction ('skating' on ice) and modified gravitation (moon walking). Fig.1 shows an example of a simulation where a walk evolved in a robot with its right leg restrained, and Fig.2 shows an example of an evolved jump from a reduced gravity run. Further details of these experiments are given in (Eaton and Davitt, 2006; Eaton and Davitt, 2007; Eaton, 2007).



Figure 1. QRIO-like robot walking with right leg restrained



Figure 2. An example of an evolved jump from a reduced gravity run

4. The importance of benchmarking

An important task currently facing researchers in the field of advanced autonomous robots is the provision of common benchmarks for performance evaluation. Current benchmarks, while useful, have their problems. A current de facto standard in this field is the RoboCup annual challenge. RoboCup operates in four categories: simulated teams, a small size league, a middle size league, and legged robots. An example small size robot is Khepera; a typical middle sized robot is the Pioneer platform, and the Sony artificial dog fits in the third category. There is also a humanoid league which of course is of specific interest. Individual skills to be mastered include navigation and localisation on the field of play, and the selection of optimal paths. Inter-individual skills include the coordination of movements with playing partners in order to pass accurately. At the top level the tasks of strategy generation and recognition of opponents' strategies are crucial.

Criticisms of RoboCup stem from the controlled environment in which the robots operate, and the fact that soccer-playing skills are quite specific and may lead to the development of highly focused robots of little use for any other task. Also, the self-localisation problem is considerably simplified by the use of highly artificial landmarks.

We recommend, therefore, the provision of a set of specifically designed experimental frameworks, and involving tasks of increasing complexity, rigorously defined to facilitate experimental reproducibility and verification. Steps are being taken in this direction, for example the European Union RoSta (Robot Standards) initiative, however we recommend the acceleration of these efforts for the provision of a universally acceptable set of benchmarks that can be used by robotics developers. By allowing easier comparisons of results from different laboratories worldwide this will facilitate important new developments in the field of intelligent autonomous robots, and humanoid robots in particular. For a further discussions of this important topic see (Gat,1995) and (Eaton et al, 2001).

5. The Bioloid robotic platform

To implement our simulated robots in the real world we currently use the Bioloid robot platform which is produced by Robotis Inc., Korea. This platform consists of a CPU, a number of senso-motoric actuators, and a large number of universal frame construction pieces.

Using this platform it is possible to construct a wide variety of robots, from simple wheeled robots to complex humanoid robots with many degrees of freedom. In addition, because of the ability to construct a range of robots with slightly different morphologies it lends itself well to evolutionary robotics experiments in which both robot "body and brain" are evolved. We initially constructed a "puppy-bot" with this kit (Fig. 3) which can walk, avoid obstacles and perform several cute tricks. With this experience we then constructed the Bioloid humanoid robot which has 18 degrees of freedom in total. A modified version of this humanoid robot was used for Humanoid Team Humboldt in the RoboCup competitions in Bremen 2006. (Hild et. al 2006) Two pieces of software are provided with the Bioloid system; the behaviour control programmer, and the motion editor (Fig.4) The behaviour control programmer programs the humanoids response to different environmental stimuli, while the motion editor describes individual motions based on the keyframe concept.

It is interesting to note the different reactions of people to the "puppy-bot" and the humanoid robot respectively. Both have quite an impressive repertoire of built in behaviours and motions, which we were able to add to. The puppy-bot can stand up, walk on four legs, avoid obstacles and, in a real attention-grabber, stand on its head (the "ears" providing suitable support) and wave its feet in the air. The humanoid on the other hand can clap its hands in response to a human clapping, walk (badly), do a little dance, get up from lying down, and avoid obstacles of a certain height.

Both repertoires are impressive on paper, but one would, perhaps, imagine the humanoid robot to elicit more amazement, it being more complex (18DOF as opposed to 15DOF) and the fact that it is in the general shape of a human. The opposite turns out to be the case. The humanoid certainly impresses, as shown by intense silence punctuated by surprised gasps. However the puppy-bot has the ability to consistently evoke a reaction of unbridled amazement and glee with its antics. Part of the answer, perhaps, comes from the level of expectation. We do not expect much of a puppy except to waddle around and act cute. And the puppy-bot does this in spades. Human behaviour, however, generally tends to be considerably more sophisticated. So when a humanoid robot walks, we do not consider what a difficult task this was to program (or perhaps evolve), but how inefficient it appears compared to human walking. And so on with other behaviours and abilities. Perhaps another factor is the popular media portraying CGI robots doing amazing tricks, beside which our real robots cannot compete (yet). And perhaps the third issue is a certain element of latent fear. The robot looks (marginally) like us, can locomote somewhat like us - in 10 years time will it be looking to take over our jobs?



Figure 3. The puppy-bot (left) and Bioloid humanoid robot (right)

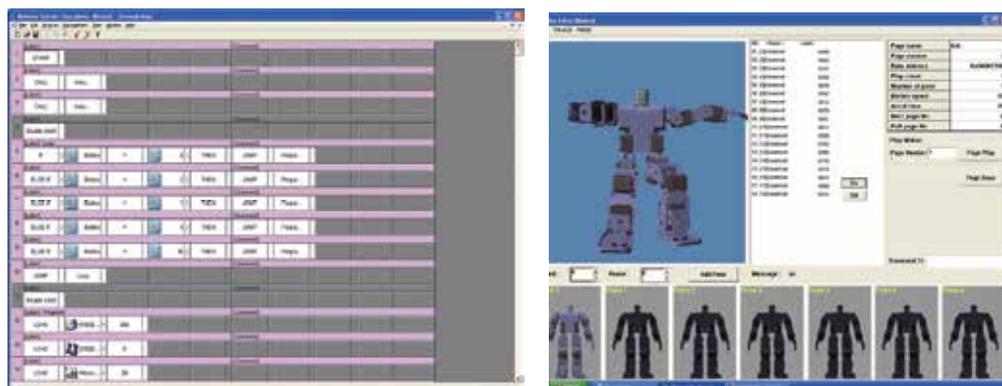


Figure 4. On the right the Bioloid Motion Editor with 18-DOF Bioloid humanoid. The current pose is shown in the main box, the pose sequence (up to six poses) is shown in the lower boxes, the Behaviour Control Programmer is shown on the left

6. The creeping mechanisation of society - a philosophical aside

As AI researchers strive, on one hand, to recreate human-like intelligence in machine form on the other hand people are being coerced and cajoled into acting and thinking in an increasingly mechanised fashion. While machines may indeed one day achieve human-level intelligence (or beyond), however we define this, the opposite will never be the case. Indeed the gap can only get wider as machinery and its computing ability becomes ever more sophisticated. There are those who argue that there is no real need for concern. By removing the drudgery from many everyday tasks (travelling to work, washing clothes etc.) are these machines, in one sense, not acting in exactly the opposite fashion. In any case, they may argue (jokingly or not) that perhaps we are seeing the nascent genesis of a new breed of intelligence; "*Human mark 2.0*" as our natural descendants, as evidenced by the recent rapid advances in the field of humanoid robotics, and indeed in evolutionary humanoid robotics as we discuss in this article.

And here lies the nexus of the problem as I see it. The potential damage may not be so much material (ever increasing workloads, tighter schedules, living and working in box-like conditions, communications becoming ever more distant and impersonal, etc.) as psychological. In the 'battle' between humans achieving machine-like standards of dexterity and speed, and machines achieving human-like intelligence there can only be one winner. Why do so many people nowadays turn increasingly to escape from the pressures of "everyday" life (which only 150 years ago would have been far from "everyday") to alcohol and other drugs? What is the reason for the recent upsurge in depression and associated ills in men, especially young men in the advanced societies? For example, in Japan, arguably the most advanced technological nation on the planet, increasing numbers of young men are turning "*hikikomori*" (literally "pulling away"), where a significant proportion (by some estimates up to 1% of the entire Japanese population) are withdrawing completely from society and locking themselves away in their room for several months, and even years at a time.

Perhaps deep down they feel, or have an inkling of, an ongoing battle in which they have no control, in which there can be only one winner, and in which they think they will feel ever more redundant. At least human females (of child-bearing age) can currently perform one

task which is impossible for machines: that is to produce the next generation of humanity; for men their spheres of dominance over machines in so many areas is being gradually whittled away with no end in sight. Like the native Americans or the Aboriginal races in Australia, when faced with what seems an impossible future- many simply give up.

Fear is insidious. Fear is pervasive. And the most insidious and pervasive fear of all is fear of the unknown, or in this case fear from an unrecognised source. It is now time for us to address these issues squarely. Rather than blindly embracing future technologies for technologies' sake we should be more critical as to their potential future benefits and drawbacks for humanity as a whole . There may well be significant social implications but the consequences of inaction could well have far reaching consequences for us all.

7. Evolving bipedal locomotion in a real humanoid robot

Returning to the main theme of this chapter, we have now constructed an accurate model in Webots of the Bioloid humanoid robot (Fig.3, right) in order to test our techniques in the real world. As in the case for the simulated experiments we use a genetic algorithm to evolve the positions of the joints of the robot at four points in the walk cycle. An existing interpolation function fills in the joint values between these keyframes and the cycle repeats until the robot falls over or until a set time limit is reached. The fitness function used is again a function based mainly on the total distance travelled by the robot, this value being doubled if the robot finishes ahead of where it started. An additional bias was added for these experiments for walking in the forward direction, by adding a value directly proportional to the distance travelled forward (rather than any other direction) to the original computed fitness. This was to counteract the robot travelling (albeit quite effectively) in a sideways direction, a pattern exacerbated by the increased stability of the Bioloid robot as opposed to its QRIO-like counterpart. Because of its importance the field specifying the maximum joint ranges is increased from 8 to 16 bits. Also, an additional four 16-bit fields define the speed of movement for each of the four keyframes. This brings the total genome length to 400 bits (two of the 20 motor fields are left unused). The population size is correspondingly increased from 80 to 100 to accommodate this increase in genome length. While each robot starts its run from the same fixed position , it inherits the initial values for its joints from the final position of the joints in the previously evaluated robot; the first robot of the following generation, which was guaranteed safe passage by the elitist mechanism, inherits the starting positions of its joints from the final position of the joints of the last robot of the previous generation. This ensures a degree of robustness in the walks generated by the evolutionary process; also because of this the maximum fitness values from generation to generation can fall as well as rise. Our previous work involved evolving a subset of the Bioloid robots' joints (Eaton, 2007), however a recent upgrade of the Webots software allowing for the detection of internal collisions has allowed us to extend the evolution to the full 18 joints of the Bioloid humanoid.

Fig.5 shows the maximum and average fitness together with the allowed joint range averaged over three runs for the 18-DOF simulated Bioloid humanoid. We see that around generation 150 the maximum fitness begins an approximately linear increase as walking patterns develop; after generation 250 this corresponds to quite a stable walk in the forward direction. Fig.6 shows a walk evolved in the Webots simulator and Fig.7 demonstrates this walk as transferred to the Bioloid humanoid. This walk corresponds to a quite rapid but slightly unstable walk in the forward direction. The transfer to the real robot is not perfect due to some small inconsistencies between the Webots model and the actual robot

indicating work remains to be done to fully "cross the reality gap" (Lipson et al., 2006) but our current results are very promising.

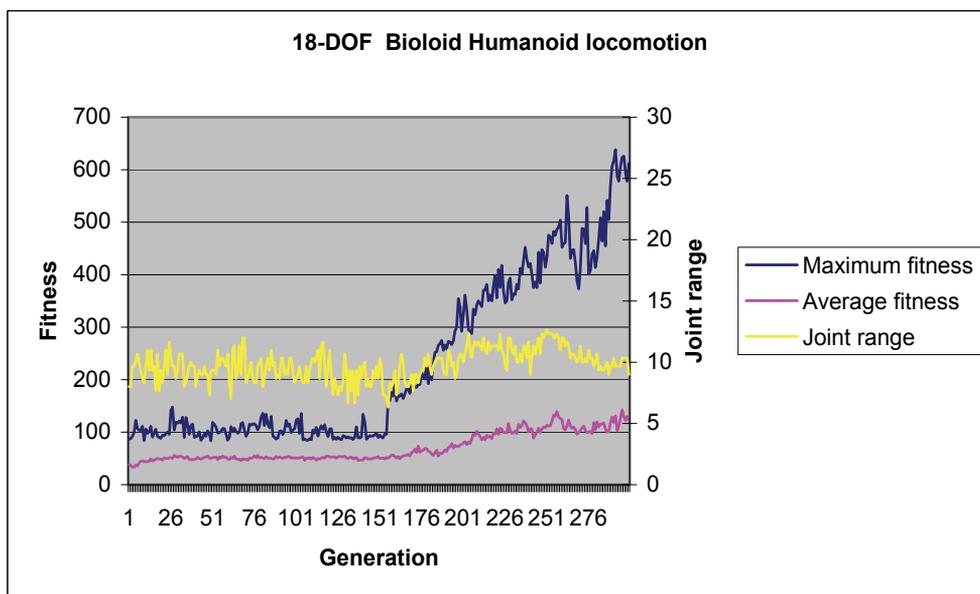


Figure 5. Maximum fitness, average fitness and joint range averaged over 3 runs for the 18-DOF Bioloid humanoid

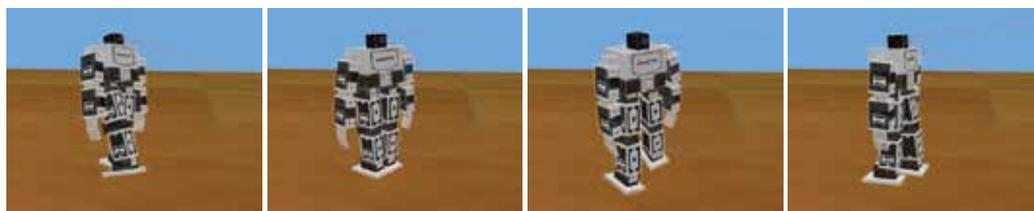


Figure 6. An evolved walk for the Bioloid humanoid



Figure 7. Two keyframe values for the evolved walk as transferred to the real robot

8. Discussion and conclusions

Following an initial introduction to the twin topics of evolutionary robotics and humanoid robotics, we discussed their recent convergence in the new field of evolutionary humanoid robotics. After presenting a survey of recent work in this field by other researchers, we introduced our own work in the area of evolving bipedal locomotion in a simulated high-DOF humanoid. After a brief discussion on the important topic of benchmarking future mobile robot performances we introduced our current hardware platform, and the implementation of locomotion, evolved in simulation using the Webots simulator, on the Bioloid platform for an 18-DOF humanoid robot. Advantages of our approach include its adaptability and the ability to generate life-like behaviours without the provision of detailed domain knowledge. While certain questions remain to be addressed, including the potential scalability of this approach to the generation of highly complex behaviours, the field of evolutionary humanoid robotics should prove a useful and powerful tool for future designers of humanoid robots.

9. Acknowledgments

Acknowledgements are due to T.J Davitt for his work on the initial version of the code for this system, and also to Diarmuid Scannell for his work on the Webots Bioloid humanoid model. Also many thanks to Olivier Michel of Cyberbotics for his swift and helpful support for my Webots queries.

8. References

- Akachi K, Kaneko K, Kanehira N et al(2005) Development of humanoid robot HRP-3P in *Proc. of 2005 5th IEEE-RAS International Conference on humanoid robots*, pp.50-55.
- Ambrose R , Ambrose C(2004) Primate anatomy, kinematics, and principles for humanoid design in *International Journal of Humanoid Robotics*, Vol.1, No. 1 (2004) pp.175-198
- Aydemir D, Iba H (2006) Evolutionary behavior acquisition for humanoid robots in *T.P. Runarsson et al.(Eds. : PPS IX, LNCS 4193, pp. 651-660, 2006.*
- Boeing A, Hanham S, Braunl T (2004)Evolving autonomous biped control from simulation to reality in *Proc. 2nd International Conference on Autonomous Robots and Agents* December 13-15, 2004, pp. 440-445.
- Bongard J, Paul C, (2001) Making evolution an offer it can't refuse: morphology and the extradimensional bypass In J. Keleman and P. Sosik (Eds.): *Ecal 2001*, LNAI 2159, pp. 401-412, 2001.
- Bongard J, Zykov V, Lipson H(2006) Resilient machines through continuous self modeling, *Science* Vol. 314, Nov. 2006, pp. 1118-1121.
- Brooks R (1997) Evolutionary robotics: where from and where to in *Evolutionary Robotics: From intelligent robots to artificial life(ER'97)* , Tokyo, Japan, pp.1-13.
- Brooks R, Braezel C, Marjanovic M et al (1998) The Cog project: building a humanoid robot. In C. Nehaniv, ed., *Computation for metaphors, analogy and agents*, Vol 1562 Springer-Verlag LNAI 1998.
- Brooks R (2002) Humanoid robots *Communications of the ACM*, March 2002, Vol.45. No.3, pp.33-38.

- Brooks R, Aryananda L, Edsinger A, Fitzpatrick P, Kemp C, O'Reilly U, Torres-Jara E, Varshavskaya P, Weber J (2004) Sensing and manipulating built-for-human environments, *International Journal of Humanoid Robotics*, Vol.1 No.1 pp.1-28.
- Craighead J, Murphy R, Burke J, Goldiez B (2007) A survey of commercial and open source unmanned vehicle simulators in *Proc. 2007 International conference on robotics and automation (ICRA)*, Arpil 2007, pp.852-857.
- Eaton M, Collins JJ, Sheehan L (2001), Towards a benchmarking framework for research into bio-inspired hardware-software artefacts, in *Artificial Life and Robotics*, Vol. 5 No. 1, 2001 pp. 40-46.
- Eaton M, Davitt T.J.(2006) Automatic evolution of bipedal locomotion in a simulated humanoid robot with many degrees of freedom in *Proceedings of the 11th International Symposium on Artificial Life and Robotics*, Jan 23-25,2006, Japan, pp. 448-451.
- Eaton M, Davitt T J (2007) Evolutionary control of bipedal locomotion in a high degree-of-freedom humanoid robot: first steps, in *Artificial Life and Robotics*, Vol. 11, No.1 2007, pp. 112-115.
- Eaton M., (2007) Evolutionary humanoid robotics : past, present and future, in Pfeifer R., Bongard J.B., Lungarella (Eds.) *50 Years of Artificial Intelligence*, Festschrift, LNAI 4850, pp. 42-53.
- Endo K, Maeno T, Kitano H,(2002), Co-evolution of morphology and walking pattern of biped humanoid robot using evolutionary computation-consideration of characteristic of the servomotors- In *Proceedings of the 2002 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, Lausanne, Switzerland, Oct. 2002, pp. 2678-2683.
- Endo K, Maeno T, Kitano H (2003) Co-evolution of morphology and walking pattern of biped humanoid robot using evolutionary computation- evolutionary designing method and its evaluation In *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, Nevada, USA, Oct. 2003, pp. 340-345.
- Endo K, Yamasaki F, Maeno T, Kitano H (2003), Co-evolution of morphology and controller for biped humanoid robot in *RoboCup 2002 : Robot soccer worldcup VI*, Springer-Verlag LNCS 2752, pp. 327-341.
- Floreano D, Urzelai, J (2000) Evolutionary Robotics: The Next Generation. In: Gomi T (ed), *Evolutionary Robotics*. III, Ontario (Canada): AAI Books.
- Full R(2001) Using biological inspiration to build artificial life that locomotes in *Evolutionary robotics: from intelligent robotics to artificial life Proceedings ER 2001 Tokyo*, Japan, Oct 2001, pp. 110-120.
- Gat E (1995) Towards a principled experimental study of autonomous mobile robots, *Autonomous Robots*, 2:179-189.
- Harvey I (2001) Artificial evolution: a continuing SAGA in *Evolutionary robotics: from intelligent robotics to artificial life Proceedings ER 2001 Tokyo*, Japan, Oct 2001, pp. 94-109.
- Hild M, Jungel M, Spranger M(2006) Humanoid team Humboldt team description 2006 in *Proceedings CD Robocup 2006*, Springer-Verlag, Bremen, Germany, June 2006.
- Ishiguro A, Fujii A, Eggenburger H, (2003), Neuromodulated control of bipedal locomotion using a polymorphic CPG circuit, *Adaptive Behavior* 2003, Vol 11(1):pp. 7-18.
- Lipson H, Bongard J, Zykov V, Malone E (2006) Evolutionary robotics for legged machines: from simulation to physical reality, Arai, T. et al (eds.), *Intelligent Autonomous Systems 9 (IAS-9)*, pp.11-18.

- Mataric M, Cliff D (1996) Challenges in evolving controllers for physical robots , *Robotics and autonomous systems* 19(1): pp.67-83.
- MacDorman K.F (2005) Mortality salience and the uncanny valley in *Proceedings of 2005 IEEE-RAS International Conference on Humanoid Robots*.
- Michel O. (2004) Webots: Professional Mobile Robot Simulation, *International Journal of Advanced Robotic Systems*, Vol. 1, No. 1, pages 39-42.
- Miyashita K, Ok S, Kazunori H (2003) Evolutionary generation of human-like bipedal locomotion, *Mechanotronics* 13 (2003) pp. 791-807.
- Mojon S (2003) Realization of a Physics Simulation for a Biped Robot Semester project, School of Computer and Communication Sciences EPFL, Switzerland 2003.
- Mori M. (1970) Bukimi no tani [the uncanny valley]. *Energy*, 7:33-35,1970.
- Nolfi S, Floreano D. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. (MIT press, 2000).
- Pfeifer R, Bongard J, How the body shapes the way we think: a new view of intelligence, MIT press, 2007
- Pollack J, Lipson H, Funes P et al (2001) First three generations of evolved robots in *Evolutionary robotics: from intelligent robotics to artificial life Proceedings ER 2001* Tokyo, Japan, Oct 2001, pp. 62-71.
- Reil T, Husbands P,(2002) Evolution of central pattern generators for bipedal walking in a real-time physics environment, In *IEEE Trans. On Evolutionary Computation*, Vol 6, No. 2, April 2002, pp. 159-168.
- Sellers W.I., Dennis L.A., Crompton R.H.(2003), Predicting the metabolic energy costs of bipedalism using evolutionary robotics, *The Journal of Experimental Biology* 206, pp. 1127-1136.
- Sellers W I, Dennis L A, Wang W J, Crompton R H (2004) Evaluating alternative gait strategies using evolutionary robotics *J.Anat* (2004) 204, pp.343-351.
- Sims K. (1994) Evolving virtual creatures , *SIGGRAPH '94* pp. 15-22.
- Wolff K and Nordin P (2002), Evolution of efficient gait with an autonomous biped robot using visual feedback In *Proc. Of the 8th Mechatronics Forum International Conference 2002*, University of Twente, Netherlands, pp.504-513.
- Wolff K and Nordin P (2003) Learning biped locomotion from first principles on a simulated humanoid robot using linear genetic programming In *Proc. Genetic and Evolutionary Computation Conference GECCO 2003*, pp. 495-506.
- Xie M, Weng J, Fontanie J (2004) Editorial, in *International Journal of Humanoid Robotics*, Vol.1, No. 1 (2004
- Yanase Y, Iba H (2006) Evolutionary motion design for humanoid robots in *GECCO '06*, July 2006, Seattle, USA.
- Zhang R, Vadakkepat(2003), An evolutionary algorithm for trajectory based gait generation of biped robot, In *Proceedings of the International Conference on Computational Intelligence, Robotics and Autonomous Systems*, Singapore 2003.
- Zykov V, Bongard J, Lipson H (2004) Evolving Dynamic Gaits on a Physical Robot *Proceedings of Genetic and Evolutionary Computation Conference*, Late Breaking Paper, GECCO'04, 2004

Real-Time Evolutionary Algorithms for Constrained Predictive Control

Mario Luca Fravolini, Antonio Ficola and Michele La Cava
*Dipartimento di Ingegneria Elettronica e dell'Informazione, Università di Perugia
Italy*

1. Introduction

In the last years, thanks to the great advancements in computing technology, Evolutionary Algorithms (EA) have been proposed with remarkable results as robust optimisation tools for the solution of complex real-time optimisation problems. In this chapter we review the most important results of our research studies concerning the design of EA-based schemes suitable for real-time optimisation problems for Nonlinear Model Based Predictive Control (MBPC). In the first part of the chapter it will be discussed some modifications of a standard EA in order to address some real-time implementation issues.

The proposed extension concerns the adoption of a new realtime adaptive mutation range to generate smooth commands, and the adoption of an intermittent feedback to face the computational delay problem. It will be shown that the main advantage of the improved technique is that it allows an effective real-time implementation of the Evolutionary-MBPC with a limited computing power. The real-time feasibility of the proposed improved Evolutionary-MBPC will be demonstrated by showing the experimental results of the proposed method applied to the control of a laboratory flexible mechanical system characterized by fast dynamics and a very small structural damping.

Later, the application of real-time EAs is extended to real-time motion planning problems for robotic systems with constraints either on input and state variables. Basing on a finite horizon prediction of the future evolution of the robot dynamics, the proposed EA-based device online preshapes the reference trajectory, minimizing a multi-objective cost function. The shaped reference is updated at discrete time intervals taking into account the full nonlinear robot dynamics, input and state constraints. A specialized Evolutionary Algorithm is employed as search tool for the online computation of a sub-optimal reference trajectory in the discretized space of the control alternatives. The effectiveness of the proposed method and the online computational burden are analyzed numerically in two significant robotic control problems.

2. Improved evolutionary predictive controllers for real-time application

Part of the following article has been previously published in: M.L. Fravolini, A. Ficola, M. La Cava, "Improved Evolutionary Predictive Controllers for real time application", Control and Intelligent Systems, vol. 31, no.1, pp.16-29,2003, Acta Press, Calgary Canada, ISSN: 1480-1752(201) .

The employment of a model of a dynamical system to plan in real-time optimal control policies is a very attractive approach for the generation of sophisticated and effective control laws. The Model Based Predictive Control (MBPC) is based on this idea. In the last years, much progress has been made toward the development of the stability theory of nonlinear MBPC (Maine & Michalaska, 1993; Scokaert et al., 1999; Gyurkvcics, 1998), but the resulting algorithms are quite difficult to be implemented in real-time. The difficulty in the implementation of nonlinear MBPC lies in the fact that a constrained non-convex nonlinear optimization problem has to be solved on-line. Indeed, in the case of nonlinear dynamics, the optimization task is highly computationally demanding and, for this reason, the online optimization problem is an important issue in the implementation of MBPC (Camacho & Bordons, 1995). The undesirable effects caused by the numerical optimization are mainly two. The first is due to the time required by the optimization procedure to compute a solution; this time is often much longer than the sampling time, thus introducing a significant delay in the control loop. The second effect is related to the fact that, in general, a numerical algorithm for nonlinear constrained MBPC cannot guarantee the optimality of the solution.

The complexity of the online optimization problem is strictly related to the structure of the nonlinear dynamics and also to the number of the decision variables and to the constraints involved. Many approaches have been proposed to face the online optimization task in nonlinear MBPC; a widely applied approach consists of the successive linearization of the dynamics at each time step and of the use of standard linear predictive control tools to derive the control policies, as proposed in (Mutha et al., 1997; Ronco et al. 1999). Although these methods greatly reduce the computational burden, the effect of the linearization could generate poor results when applied to the real nonlinear system. Other methods utilize iterative optimization procedures, as Gradient based algorithms or Sequential Quadratic Programming techniques (Song & Koivo 1999). These methods, although can fully take into account the nonlinear dynamics of the system, suffer from the well known problem related to local minima. A partial remedy is to use these methods in conjunction with a grid search as proposed in (Fischer et al., 1998). Another approach is represented by discrete search techniques as Dynamic Programming (Luus, 1990) or Branch and Bound (Roubos et al. 1999) methods. In this approach the space of the control inputs is discretized and a smart search algorithm is used to find a sub optimum in this space. The main advantages of search algorithms are that they can be applied regardless of the structure of the model; furthermore, the objective function can be freely specified and it is not restricted to be quadratic. A limitation of these methods is the fast increase of the algorithm complexity with the number of decision variables and grid points, known as "the curse of the dimensionality".

In the last years, another class of search techniques, called Evolutionary Algorithms (EAs), showed to be quite effective in the solution of difficult optimal control problems (Foegel, 1994; Goldberg 1989). Although at the beginning the largest part of the EA applications were developed for offline optimization problems, recently, thanks to the great advancements in computing technology, many authors have applied with remarkable results EAs for online performance optimization in the area of nonlinear and optimal control. Porter and Passino showed clearly that Genetic Algorithms (GA) can be applied effectively either in adaptive and supervisory control (Porter & Passino, 1998; Lennon & Passino 1999) or as nonlinear observers (Porter & Passino 1995). Liaw (Liaw & Huang 1998) proposed a GA for online

learning for friction compensation in robotic systems. In references (Fravolini et Al. 1999 (a); Fravolini et Al. (b)) the authors proposed to apply EAs for the online trajectory shaping for the reduction of the vibrations in fast flexible mechanical systems.

Recently, some authors have pointed out the possibility of applying EAs as performance optimizer in MBPC. Onnen in (Onnen et Al., 1999) proposed a specialized GA for the optimization of the control sequence in the field of process control. He showed the superiority of the evolutionary search method in comparison to a branch-and-bound discrete search algorithm. Similar algorithms have been also proposed in (Martinez et Al. 1998; Shin & Park, 1998).

Although very interesting from a theoretical point of view, all these studies were carried out only by means of simulations and scarce attention was paid to computational and real-time implementation issues; furthermore, to the best of our knowledge, no practical application of this technique has been reported in literature. In the first part of this paper, therefore, we discuss in details some practical real-time implementation issues of the evolutionary MBPC that have not been discussed in (Onnen et Al., 1999; Martinez et Al. 1998; Shin & Park, 1998). In fact, we experimented that the application the algorithm [19] could generate practical problems. As noticed also in (Roubos et Al. 1999), the suboptimal command signal often exhibits excessive chattering that cause unnecessary solicitations of the actuation system. To cope with this problem we propose to improve the basic algorithm (Linkens, 1995) by applying an online adaptation of the search space by defining a new adaptive mutation operator. We will show that smoother solutions can be obtained without significant degradation of the performance. A second aspect that should be faced is the computational delay that originates when computational intensive algorithms, as EAs, are applied for real-time optimization. This aspect is particularly important in the case of a limited computational power and fast sampling rates. To overcome this problem we modified the basic algorithm by inserting an intermittent feedback strategy. The advantage of this strategy is the possibility of decoupling the computing time by the system sampling time; this makes the EA based optimization procedure more flexible than the basic one for real-time implementation.

Another aspect that has not been discussed in previous works is the analysis of the repeatability of the control action of the EA based MBPC; this aspect is strictly related to the reliability of the proposed strategy. This issue has been addressed by means of a stochastic analysis. The first part of the paper terminates with a comparison of the performance provided by the improved EAs and an iterative gradient-based optimization procedure. In the second part of the paper we report the experimental results obtained by implementing the improved algorithm for the real-time MBPC of a nonlinear flexible fast-dynamic mechanical system.

3. Nonlinear Predictive Control

Model Based Predictive Control includes a wide class of control strategies in which the control law is derived on the basis of the estimation of the future evolution of the system predicted by a model. A nonlinear MBPC is based on the following three main concepts (Garcia et Al. 1989; Levine, 1996):

- A) the use of a model to predict the future evolution of the real system;
- B) the online computation of the optimal control sequence, by minimizing a defined index of performance that quantifies the desired behavior;

C) the receding horizon: only the first value of the optimal sequence is applied; then the horizon is shifted one sample in the future and a new sequence is recalculated.

One of the main advantages of the MBPC is the possibility to handle general constraints either on the inputs or on state variables. A block diagram of a MBPC is shown in Fig. 1.

- *The Model and the System:* In MBPC the model of the system is expressly used to compute the control law; for this reason the model should be as accurate as possible. There is no restriction on the structure of the model. If complete physical insight on the system is available, the model could be analytically expressed in state space form; on the other hand, if only a scarce knowledge is available, a black box model can be employed. In the last case, the system is often implemented by a neural network or a fuzzy system.

Remark 1: In most of the applications, the system block in Fig. 1 represents the open loop dynamics of the nonlinear systems; however in some applications it can comprise either a preexistent inner control loop or an expressly designed feedback controller. In the first case as in (Shiller & Chang, 1995) the MBPC is employed to improve the performance of the existing feedback controller. In the second case as in (Fravolini et Al., 2000) an inner feedback controller is designed in order to directly compensate the effects that are difficult to be included in system prediction model, as friction and stiction.

- *The Optimal Control Law:* In MBPC the optimal command sequence $u^*(k)$ is determined as the result of an optimization problem at each sampling instant. All the performance specifications are quantified by means of a cost index. An index, which is often employed, is the following one:

$$J = \sum_{j=N_1}^{N_2} p_k \|\hat{y}(k+j|k) - y_d(k+j)\|^2 + \sum_{j=1}^{N_u} q_k \|\Delta u(k+j-1)\|^2 + J_1 \quad (1)$$

The first term of J evaluates the square error between the desired future output signal $y_d(k+j)$ and the predicted future output $\hat{y}(k+j|k)$, estimated on the basis of the prediction model and the feedback information available at instant k . This error is evaluated over a defined *prediction horizon*, which is a window of $N_2 - N_1$ samples. The second term of J is the control contribution over the *control horizon* window of N_u samples, where ($\Delta u(k) = u(k) - u(k-1)$). Coefficients p_k and q_k weights the different terms of J. The term J_1 is a further cost function that can be used to take into account other specifications or constraints. MBPC is mainly used in conjunction with constraints on the amplitude and the rate of variation of the input and output variables, namely:

$$U^- < u(k) < U^+ \quad \Delta U^- < \Delta u(k) < \Delta U^+ \quad (2)$$

$$Y^- < y(k) < Y^+ \quad \Delta Y^- < \Delta y(k) < \Delta Y^+ \quad (3)$$

The fulfillment of the constraints (2) are required to take into account the practical limitations of the actuation system, while constraints (3) can prevent the system to work in undesirable regions. The minimization of index J is performed with respect to the

decision variables, which is the sequence of the control increments: $[\Delta u(k), \Delta u(k+1), \dots, \Delta u(k+N_u-1)]$.

- *The receding horizon and the feedback filter:* In the receding horizon strategy only the first sample of the optimal sequence $u^*(k+)$ is applied to the system; subsequently, the horizon is shifted one step in the future and a new optimization is repeated on the basis of the measured feedback information. In the basic form, the horizons are shifted at each sampling instant. Although this strategy introduces feedback information in the control loop at the sampling rate, this can require a large computational power in the online implementation; on the other hand (see section 6), it is possible to decrease the computational burden by applying an intermittent feedback (Ronco et AL., 1999). The structure and complexity of the feedback filter depends on the strategy employed to insert the feedback information in the optimization process. In the simplest case the filter is low-pass and is used only to reduce the measurement noise. This approach is often used in conjunction to input-output models where the measured output is mainly used to recover steady state errors. In case of state space models, the feedback filter could be more complex: it can be either a bank of filters used to estimate derivatives or integrals, or a model based nonlinear observer. In the last case the possibility to recover the system states allows periodic system/model realignment. The realignment prevents excessive drift of the prediction error when a long-range prediction is required.

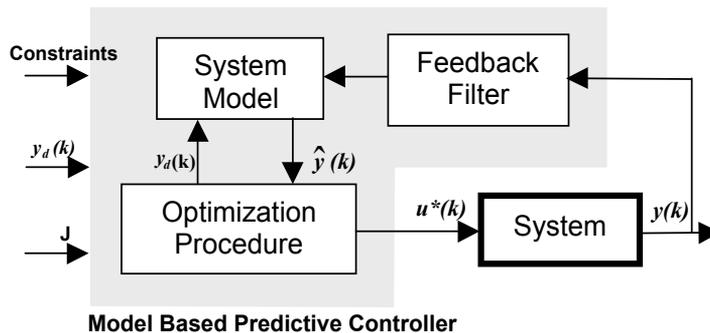


Figure 1. Nonlinear Model Predictive control

4. The benchmark laboratory nonlinear system

The performance of the improved Evolutionary MBPC has been experimentally tested on a nonlinear laboratory flexible mechanical system. The system is composed of a flexible beam clamped at one end; a controlled pendulum is hinged on the free tip and is used to damp out the beam oscillations; the motion occurs in the horizontal plane. A DC motor is employed to regulate the angular position of the pendulum in order to damp out the vibrations on the flexible beam. Fig. 2 shows the experimental set-up. The system is characterized by a very small structural damping and its stabilization in a short time represents a significant test because of the fast dynamics and the presence of oscillatory modes that require either an accurate model or a short sampling time or a long prediction horizon. This situation is known to be critical in the implementation of MBPC algorithms.

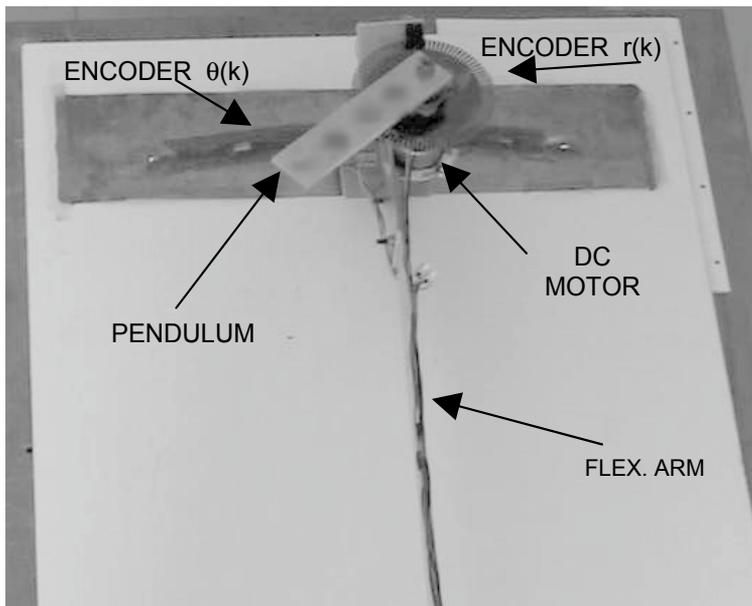


Figure 2. The laboratory benchmark flexible systems

Fig. 3a shows the mechanical model of the system, the main parameters of which are reported in table 1. The system is equipped with optical encoders that allow the measurement of the tip deflection r and the angle ϕ of the pendulum (since the beam deflection is small we exploited the relation $r \approx L \cdot \theta$). Because of the most of the vibration energy is associated to the first vibration mode of the beam, its dynamics can be described with a sufficient accuracy by a mass-spring-damper mechanical system (Fig. 3b). The equations of motion of a flexible mechanical system moving in a horizontal plane are of the form:

$$\mathbf{B}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{Q} \quad (4)$$

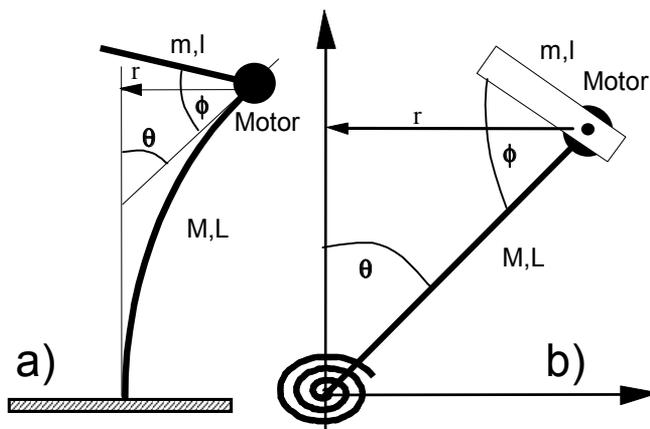


Figure 3. The structure (a), and its model (b)

M	0.689 kg	Mass of the beam
m	0.070 kg	Mass of the pendulum
L	1.000 m	Length of the beam
l	0.086 m	Length of the pendulum
K_1	25.3 Nm/rad	Elastic coeff. of the beam
C_1	0.008 Nm/s rad	Damping coeff. of the beam
C_2	0.050 Nm/s rad	Damping coeff. of the pendulum

Table 1. Parameters of the Model

where \mathbf{B} is the inertia matrix, \mathbf{C} comprises Coriolis, centrifugal, stiffness and damping terms, $\mathbf{q}=[\theta \phi]^T$ is the vector of Lagrangian coordinates and \mathbf{Q} is the generalized vector of the input forces. By applying the standard Lagrangian approach the following model matrices were obtained:

$$\mathbf{B} = \begin{bmatrix} (M+m)L^2 + ml^2 - 2mLl \cos \phi & ml^2 - mL \cos \phi \\ ml^2 - mL \cos \phi & ml^2 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 2mL\dot{\phi}\dot{\theta} \sin \phi + mL\dot{\phi}^2 \sin \phi + K_1\theta + C_1\dot{\theta} \\ -mL\dot{\theta}^2 \sin \phi + C_2\dot{\phi} + C_{cou} \operatorname{sgn}(\dot{\phi}) \end{bmatrix} \quad (5)$$

$$\mathbf{Q} = [0 \ \tau]^T$$

where τ is the control torque and the term C_{cou} represents the Coulomb friction coefficient acting on the motor; this term cannot be modeled accurately. The inaccuracy produced by this uncertainty can generate a significant discrepancy between the simulated and the real pendulum position. A method of compensating for modeling errors that cause inaccuracies in the estimation of the pendulum angle is to redefine the model such that the input is not the motor *torque* τ , but rather the motor *position* $\phi(t)$ and its derivatives (Geniele et Al., 1997)]. In this case, using an inner position feedback controller (for instance, a PD controller), it is possible to track accurately a reference trajectory $\phi_d(t)$. If the desired trajectory is smooth, the tracking error can be very small. In this case we can assume $\phi(t) \approx \phi_d(t)$ and the variables $\phi_d(t), \dot{\phi}_d(t), \ddot{\phi}_d(t)$ are the new inputs to the model (5). By applying this strategy it is possible to recast the model equations (4-5) obtaining a reduced order model relating the motor position $\phi_d(t)$ to the beam deflection $\theta(t)$. The reduced dynamics satisfy the equation:

$$b_{11}(\phi_d)\ddot{\theta} + b_{12}(\phi_d)\ddot{\phi}_d + c_1(\theta, \phi_d, \dot{\theta}, \dot{\phi}_d) = 0 \quad (6)$$

where b_{11}, b_{12}, c_1 are the entries of matrix \mathbf{B} and vector \mathbf{C} . Equation (6) describes the nominal model used for the prediction by the MBPC.

Since a MBPC is discrete time device, it updates the outputs at a defined sampling rate T_s , the control inputs $\phi_{des}(t)$ and its derivatives are kept constant during a sampling interval $[kT_s, (k+1)T_s]$ by a zero-order hold filter. The decision variables for the optimization problem (1) are the control increments:

$$\begin{aligned} & [\Delta u(k+1), \Delta u(k+2), \dots, \Delta u(k+N_2)] = \\ & [\Delta \ddot{\phi}_{des}(k+1), \Delta \ddot{\phi}_{des}(k+2), \dots, \Delta \ddot{\phi}_{des}(k+N_2)] \end{aligned} \quad (7)$$

The desired position $\phi_d(t)$ and velocity $\dot{\phi}_d(t)$ are computed by numerical integration using the 4th order Runge Kutta method. The EA based MBPC procedures was written in C code and run in real-time on a dSPACE 1003 DSP board, based on a TMS320C40 DSP.

5. Evolutionary Algorithms

Evolutionary Algorithms are multi point search procedures that reflect the principle of evolution, natural selection, and genetics of biological systems (Goldberg, 1989; Porter & Passino 1998). An EA explores the search space by employing stochastic rules; these are designed to quickly direct the search toward the most promising regions. The EAs work with a population of solutions rather than a single point; each point is called chromosome. A chromosome represents a potential solution to the problem and comprises a string of numerical and logical variables, representing the decision variables of the optimization problem. An EA maintains a population of chromosomes and uses the genetic operators of “selection” (it represents the biological survival of the fittest ones and is quantified by a fitness measure that represents the objective function), “crossover” (which represents mating), and “mutation” (which represents the random introduction of new genetic material), to generate successive generations of populations. After some generations, due to the evolutionary driving force, the EA produces a population of high quality solutions to the optimization problem.

The implementation of an EA can be summarized by the following sequence of standard operations:

1. *(Randomly) Initialize a population of N solutions*
2. *Calculate the fitness function for each solution*
3. *Select the best solutions for reproduction*
4. *Apply crossover and mutation to selected solutions*
5. *Create the new population*
6. *Loop to step 2) until a defined stop criterion is met*

The implementation of the evolutionary operators of selection, crossover and mutation is not unique. The first and most known EA is the simple GA (Goldberg 1989), but a large number of modifications have been proposed and applied by many authors. In this paper, the attention will be focused mainly on the evolutionary operators expressly designed for MBPC. These operators were introduced in previous works (Fravolini et Al. 1999 (a); Fravolini et Al. (b)) and tested by mean of extensive simulation experiments. Some of these operators are similar to the corresponding ones reported by (Onnen, 1997; Martinez, 1998) and constitute a solid, tested and accepted base for Evolutionary MBPC.

5.1 The Standard Evolutionary MBPC operators

- *Fitness function:* The natural choice for the cost function to be minimized online is represented by the index J in (1). Since in the EA literature it is more common to refer to fitness function optimization than to cost function minimization, the following fitness function is defined:

$$f = 1/J \quad (8)$$

- *Decision variables:* The decision variables are the sequence of the future N_u input increments $\Delta u(k+j)$ in (7).
- *Chromosome structure and coding:* A chromosome is generated by the juxtaposition of the coded sequence of input increments (7). Concerning the codification of the decision variables, some alternatives are possible. Binary or decimal codifications, although used in many simulative studies as in (Onnen, 1997; Martinez, 1998), are not particularly suited in the real-time application due to the time consuming coding and decoding routines. Real and integer coded variables do not require any codification; but the evaluation of evolutionary operators for real numbers is slower than the corresponding routines working with integers; therefore, in this work a decimal codification was employed. A coded decision variable x can assume an integer value in the range $\Omega: 0, \dots, L_0, \dots, L_x$ where L_x represents the discretization accuracy; this set is uniformly mapped in the bounded interval $\Delta U^- \leq \Delta u \leq \Delta U^+$. L_0 is the integer corresponding to $\Delta u=0$; The i -th chromosome in the population at t -th generation is a vector of integer numbers $X_i^t = [x_{i,1}^t, x_{i,2}^t, \dots, x_{i,N_u}^t]$, the actual values of the corresponding decision variables $\Delta u(k)$ can be obtained by the following decoding scheme:

$$\Delta u(k+j) = \Delta U^- + \Delta U \cdot x_{i,j} \quad j = 1, \dots, N_u \quad (9)$$

where $\Delta U = |\Delta U^+ - \Delta U^-|/L_x$ is the control increment resolution. The sequence of the applied controls is:

$$u(k+j) = u(k-1+j) + \Delta u(k+j) \quad j = 1, \dots, N_u \quad (10)$$

- *Selection and Reproduction :* Selection and reproduction mechanisms act at two different levels during the on-line optimization:
 - *Selection and Reproduction within time interval k .* Since a limited computational time is available within the time interval k , it is essential to not loose the best solutions during the evolution and to use them as "hot starters" for the next generation. For this reason a *steady state* reproduction mechanism is used, namely the best S chromosomes in the current generation pass unchanged in the next one. The remaining part of the population is originated on the basis of a *rank selection* mechanism: given a population of size N , the parent solutions are selected within the mating pool of the best ranked D individuals. This approach is similar to the algorithm described in (Yao & Sethares, 1994).
 - *Heredity (between time interval k and $k+1$).* At the beginning of the $k+1$ -th time interval, since the horizon is shifted one step in the future, the genes of the best S' chromosomes are shifted back of one location within the chromosome; in this way the first values are lost and replaced by the second ones and so on. The values on the last positions are simply filled by keeping the current value. The shifted S' chromosomes represent the "hot starters" for the optimization in the $k+1$ -th time interval; the remaining $N-S'$ chromosomes are randomly generated.

- *Crossover*: Uniform crossover has been implemented. Given two selected chromosomes X_i^t and X_j^t , two corresponding variables are exchanged with a probability p_c , namely:

$$x_{i,k}^{t+1} = x_{j,k}^t \text{ and } x_{j,k}^{t+1} = x_{i,k}^t. \quad (11)$$

- *Mutation*: Mutation is applied with probability p_m to each variable of a selected chromosome X_i , according to:

$$x_{i,k}^{t+1} = x_{i,k}^t + \text{rand}(\bar{\Delta}) \quad (12)$$

where $\text{rand}(\bar{\Delta})$ is a random integer in the range $[-\Delta_x, +\Delta_x]$ and Δ_x is the maximum mutation amplitude.

- *Constraints*: During the optimization it is possible that a decision variable $x_{i,j}^t$ (due to mutation) violates the maximum or minimum allowed value in Ω ; this can be easily avoided by applying the following threshold operators:

$$\begin{cases} \text{if } x_{i,j}^t > L_x \Rightarrow x_{i,j}^t = L_x \\ \text{if } x_{i,j}^t < 0 \Rightarrow x_{i,j}^t = 0 \end{cases} \quad (13)$$

The application of (13) automatically guarantees that $\Delta U^- \leq \Delta u \leq \Delta U^+$. Similarly, the fulfillment of constraint $U^- \leq u \leq U^+$ is guaranteed by applying the thresholds:

$$\begin{cases} \text{if } u(k) + \Delta u(k) > U^+ \Rightarrow x_{i,k}^t = L_0 + \text{int}\left(\frac{U^+ - u(k)}{\Delta_U}\right) \\ \text{if } u(k) + \Delta u(k) < U^- \Rightarrow x_{i,k}^t = L_0 + \text{int}\left(\frac{U^- - u(k)}{\Delta_U}\right) \end{cases} \quad (14)$$

It is also possible to take into account other constraints as (3) by employing the penalty function strategy.

- *Termination criterion*: In a real-time application the termination criterion must take into account the hard bound given by the real time constraint; this implies that the maximum number of generations is limited by the fulfillment of this constraint. When the real-time deadline is met, the optimization is stopped, and the first decoded value of the best chromosome $\Delta u^*(k)$ is applied to the system.

6. Improved Operators for Real-Time Implementation

The following sections describe the improvements of the standard algorithm reported in section 4 that we propose to apply in the real-time implementation. Some of these operators are not new in the field of MBPC; however, the novelty consists of the adaptation of these concepts within an Evolutionary real-time optimization procedure.

6.1 Adaptation of the search space

The employment of a discrete search technique to minimize the index (1) introduces a tradeoff between the number of discrete control alternatives and the computational load required for the exploration of the discrete space. The performance of an EA, although it does not perform an exhaustive exploration of the search space, is anyway influenced by the dimensionality of the grid; this aspect is particularly relevant in real-time applications where the computing time is limited. A coarse discretization could generate unsatisfactory results as undesired oscillations of the output variable, while a too fine grid could require a prohibitive time to locate a satisfactory solution. The problems related to the excessive chattering in the control signal or in controller gains have been previously mentioned by other authors as in (Roubos et Al., 1999; Lennon & Passino, 1999). By applying a scaling factor to the size of the region in which the control alternatives are searched can significantly decrease this problem. As proposed in (Sousa & Setnes, 1999), a way to achieve this goal is to adapt the dimension of the search space depending on the predicted deviation of the output from the desired reference signal. When the prediction error keeps small within the prediction horizon, it is probably not convenient to explore the whole allowed region of the control alternatives; rather, it is opportune to concentrate the search around the origin. On the other hand, when a large error is predicted, the range of the control alternatives should be large to enable a fast recover.

In the context of evolutionary MBPC, the exploration of new regions is mainly carried out by the application of the mutation operator (12) that causes a random incremental variation in the range $\pm\Delta_x$ around the current value. For this reason, we propose to improve the basic algorithm by conveniently scaling during the real-time operation the mutation range $\bar{\Delta}$ by means of an adaptive gain $\alpha(k) \in [0,1]$. The adaptive mutation range is therefore defined as:

$$\bar{\Delta}(k) = \alpha(k) \cdot \Delta_x \quad (15)$$

Different criteria can be used to adapt the parameter $\alpha(k)$. In (Sousa, 1999) the adaptation of the parameter $\alpha(k)$ is carried out by means of a fuzzy filter on the basis of the current estimation error and the current change of error. In this work we propose to adapt the parameter in function of the predicted mean absolute output error \bar{E}^* for the best solution:

$$\bar{E}^*(k) = \frac{1}{(N_2 - N_1)} \sum_{j=N_1}^{N_2} |y_d(k+j) - \hat{y}(k+j)| \quad (16)$$

\bar{E}^* is a meaningful index because it takes into account not only the current value of the error but also the predicted behavior of the estimation error. When $\bar{E}^*(k)$ is small, $\alpha(k)$ should be small to allow a fine regulation near the steady state; otherwise, when $\bar{E}^*(k)$ is large, $\alpha(k)$ should be large to allow big corrections. The following law defines a possible scaling:

$$\alpha(k) = 1 - \exp(-\bar{E}^* / \sigma) \quad (17)$$

where $\sigma > 0$ is a parameter that regulates the adaptation speed. The proposed law has only one design parameter (σ) that is not really critical and, thus, allows for the use of some heuristics to cope with the uncertainty in its definition. By employing the adaptive law (15),

the choice of the maximum mutation amplitude Δ_x is no more critical, because the adaptive gain $\alpha(k)$ scales its range accordingly.

6.1.1 A simulation example

To show the effect of the adaptation of the search space we report the results of a simulation study performed on the model of the mechanical system (6). The decision variables is the sequence (7). To find suitable values for the setting of the MBPC parameters some trials were performed. The length of prediction horizon was chosen in order to cover a complete period of oscillation of the first mode of vibration of the beam; shorter horizons gave unsatisfactory responses, that is a very long stabilization transient. In the simulation we chose equal values for the prediction and control horizons; the final values were: $N_1=0, N_2=N_u=32$; the sampling time was $T_s=0.04s$ (this implies a horizons length of $N_u \cdot T_s = 1.28s$). The other parameters of the MBPC are reported in table 2.

Since the scope of the control system is to damp out the oscillations of the tip position $r(k)$ of the beam ($r(k) \cong L \cdot \theta(k)$) (see Fig. 3), we assumed as the desired target position the trajectory $y_d(k) = \theta_d(k) = 0$; the resulting objective function is:

$$J = \sum_{j=0}^{N_u} |\theta(k+j) - y_d(k+j)| = \sum_{j=0}^{N_u} |\theta(k+j) - 0| \quad (18)$$

$N = 20$	$D = 6$	$S = 2$	$S = 4$
$p_m = 0.1$	$p_c = 0.8$	$K = 5$	$T_s = 0.04$
$L_x = 4000$	$\Delta_x = 1000$	$\sigma = 0.001$	$N_u = 32$

Table 2. Parameters of the MBPC

It is worth noting that, although cost function (18) is different from index (1), it does not influence the functionality of the EA. In index (18), no cost is added to weight the contribution of the control effort, because it was observed in the real implementation that the most important constraint on the control signal is imposed by its rate of variation, due to the limited bandwidth of the actuator. The constraint $-8 < \Delta u(k) < 8$ allows the generation of reference signal that can be accurately tracked by actuator. Fig. 4a shows the response of the controlled system in the case of no adaptation of the mutation range; in the same figure it is also reported the uncontrolled response (dotted line). Fig. 4b shows the same response in the case that the adaptive law of the mutation range is active. While in the two cases a not significant difference in the performance variable is observed, on the other hand, a big discrepancy is present in the sequence of the control input increments (Fig. 5a-b). In the case of fixed mutation range the control increments exhibit an excessive chattering also when the system is near the steady state. The presence of persistent spikes in the control signal is due to the difficulty of the basic algorithm to select a smooth sequence of small increments in a large space of control alternatives in a limited number of generations. In the case of adaptation of the mutation range, the control increments are more reasonable. Fig. 6 a-b shows the control signals applied to the system in the two cases. Although an unavoidable chattering is present during the transitory, it completely disappears when the system is near the steady state. This example shows that the adaptive reduction of the dimensionality of

the search alternatives helps the algorithm to select a sufficiently smooth signal able to drive the mechanical system to the steady state; furthermore, the adaptive mutation range does not degrade the performance of the controlled system. For this reason the adaptive mutation ranges represent a valid improvement of the basic algorithm for real-time Evolutionary MBPC of the system under inspection. Fig. 7 shows the adaptation of the gain $\alpha(k)$ during the transitory.

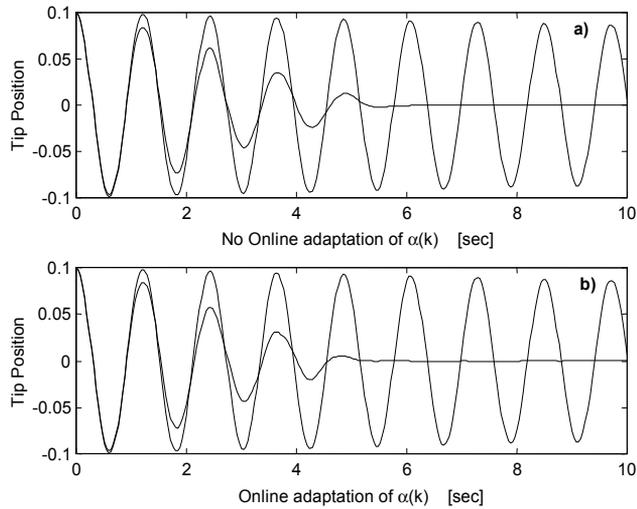


Figure 4. Tip position (controlled and free) without the adaptation of the search space (a), and with the adaptation of the search space (b)

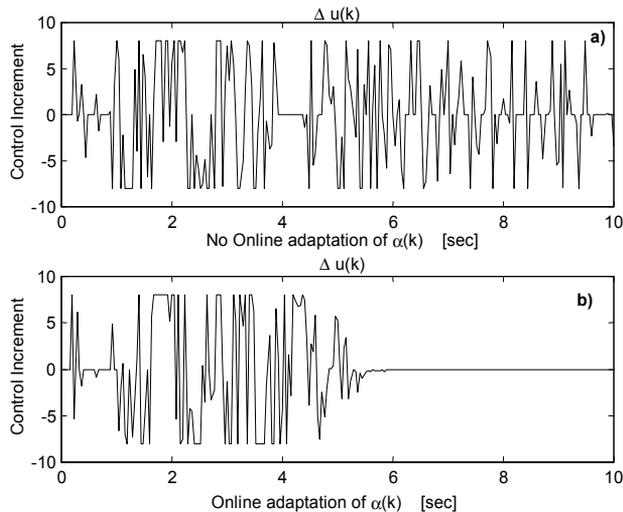


Figure 5. Control input increments without the adaptation of the search space (a), and control input increments with the adaptation of the search space (b)

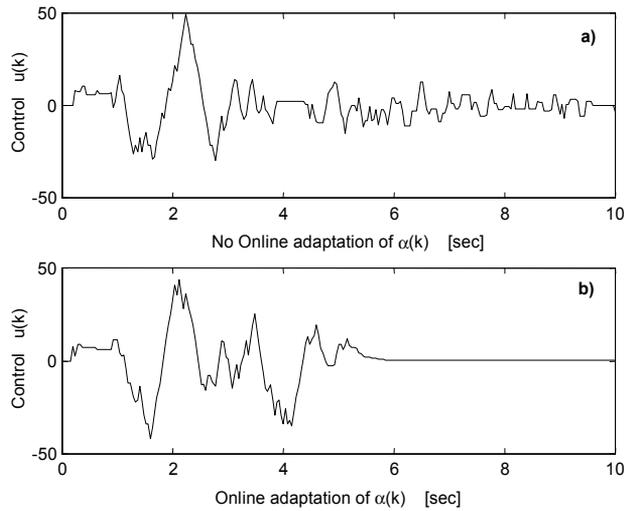


Figure 6. Control signal without the adaptation of the search space (a), control signal with the adaptation of the search space (b)

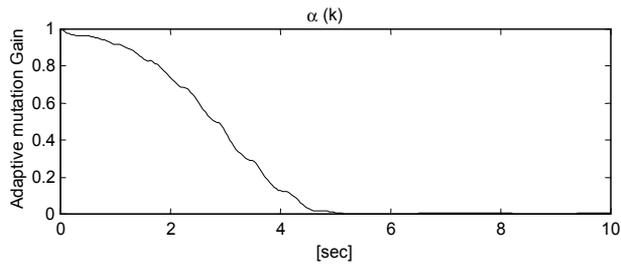


Figure 7. Online adaptation of gain $\alpha(k)$

6.2 The computational delay problem

The implementation of a MBPC procedure implies the online optimization of the cost index J at every sampling period T_s . In most of theoretical and simulation studies concerning the MBPC, the problems related to the computational delay, that is the CPU time T_c required for the numerical optimization of index (1), are seldom taken into account. In the ideal situation ($T_c=0$), the optimal control signal applied in the m -th sampling interval depends directly on the current state, $x(kT_s)$, at the same instant. Under this hypothesis the optimal control law is defined by the following function $f_s(\cdot)$:

$$u_k^*(t) = f_s(x(kT_s), u(kT_s), t) \quad t \in [mT_s, (m+1)T_s] \quad (19)$$

Although this hypothesis can be reasonable in some particular cases, the computational delay is a major issue when nonlinear systems are considered, because the solution of a nonlinear dynamic optimization problem with constraints is often computationally intensive. In fact, in many cases the computation time T_c required by the optimization

procedure could be much longer than the sampling interval T_s , making this control strategy not implementable in real-time. Without loss of generality we assume that the computing time is a multiple of the sampling time:

$$T_c = H \cdot T_s \quad (20)$$

where H is an integer. The repetition of the optimization process in each sampling instant is related to the desire of inserting robustness in the MBPC by updating the feedback information at the beginning of each sampling interval (T_s) before the new optimization is started. Generally, the mismatches between system and model cause the prediction error to increase with the prediction length and for this reason the feedback information should be exploited to realign the model toward the system to keep the prediction error bounded. The simplest strategy to take into account the system/model mismatches is to use a parallel model and to add the current output model error $e(k) = y(k) - \hat{y}(k)$ to the current output estimation. In this way the improved prediction to be used in index (1) is:

$$\hat{y}(k+j) \leftarrow \hat{y}(k+j) + e(k) \quad j = N_1, \dots, N_2 \quad (21)$$

This approach works satisfactory to recover steady state errors and for this reason it is widely used in process control and in presence of slow and not oscillatory dynamics (Linkens & Nyongesa, 1995). In case a white box model of the system is available, a more effective approach is to employ the inputs and the measured outputs to reconstruct the unmeasured states of the system by means of a nonlinear observer (Chen, 2000); this allows a periodic realignment of the model toward the system.

6.2.1 Intermittent feedback

In the case the prediction model generates an accurate prediction within a defined horizon, it is not really necessary to perform the system/model realignment at the sampling rate T_s . As proposed in (Chen et Al., 2000) an intermittent realignment is sufficient to guarantee an adequate robustness to system/model mismatches. Following this approach, the effects of the computational delay are overcome by applying to the system, during the current computation interval $[kT_c, (k+1)T_c]$, not only the first value of the optimal control sequence yielded in the previous computation interval $[(k-1)T_c, kT_c]$, but also the successive $H-1$ values ($T_c = H \cdot T_s$). When the current optimization process is finished, the optimal control sequence is updated and the feedback signals are sampled and exploited to perform a new realignment; then a new optimization is started. By applying this strategy the realignment period is therefore equal to the computation time T_c . According to this approach the system is open loop controlled during two successive computational intervals and the optimal control profile during this period is defined by the following function $f_c(\cdot)$:

$$\begin{aligned} u_{k|k-1}^*(t) &= f_c \left(x(kT_c), u \left([kT_c, (k+1)T_c], t \right) \Big|_{k-1} \right); \\ t &\in [kT_c, (k+1)T_c] \end{aligned} \quad (22)$$

where $u_{k|k-1}^*(t)$ is the predicted sequence to be applied in the k -th computational interval that has been computed in the previous interval. The main advantage of the intermittent feedback strategy is that it allows the decoupling between the system sampling time T_s and

the computing time T_c ; this implies a significant decrease in the computational burden required for the real-time optimization. On the other hand, a drawback of the intermittent feedback is that it inserts a delay in the control action, because the feedback information has an effect only after T_c seconds. The evolutionary MBPC algorithms described in (Onnen et Al., 1999; Martinez et Al. 1998) do not take into account the computational delay problem; therefore, their practical real-time implementation strictly depends on the computing power of the available processor that should guarantee the execution of an adequate number of generations within a sampling interval $[mT_s, (m+1)T_s]$. The employment of an intermittent feedback strategy allows the enlargement of the computation time available for the convergence of the algorithm and makes the algorithm implementable in real-time also with no excessively powerful processors.

The choice of the computing time T_c (realignment period) represents an important design issue. This period should be chosen as a compromise between the two concurrent facts:

1. The enlargement of the computing time T_c allows to refine the degree of optimality of the solution by increasing the number of generations within an optimization period.
2. Long realignment periods cause the prediction error to increase, as a consequence of system/model mismatches.

6.2.2. Effects of the intermittent feedback

To evaluate the effects of the intermittent feedback we considered two exemplificative simulations.

Reference is made to the model (4-5) of the flexible mechanical system. We investigated the following situations:

Case A) No system/model mismatches (ideal case)

Case B) Significant modeling error (realistic case)

Since the Evolutionary MBPC optimization procedure is based on a pseudo randomized search, it is unavoidable that the repetition of the same control task generates slightly different sub optimal control sequences. For this reason the investigation of the performance should be carried out by means of a stochastic analysis by simulating a significant number of realizations (in our analysis 20 experiments of 10 seconds each). We choose as performance measure the mean and the standard deviation of the mean absolute tracking error \bar{e} for the tip position of the beam starting from the deflected position $\theta = 0.1 \text{ rad}$, $\dot{\theta} = 0$, $\phi = 0$, $\dot{\phi} = 0$. This variable is defined as:

$$\bar{e}(\xi) = \frac{1}{N} \sum_{m=1}^{T_{end}} |\theta(m, \xi) - 0| \quad (23)$$

where the (stochastic) variable ξ is used to put into evidence the stochastic nature of the variable \bar{e} .

- *Case A, no model uncertainty:* The Evolutionary MBPC is set according to the parameters of table 2. The scope of the analysis is to show the effect on the performance caused by the enlargement of the computing time. As the computing time T_c is expressed as a multiple of the sampling time T_s , its enlargement is obtained by increasing the value of the integer H in (20). The analysis is performed by varying the number of the generations K that are computed during T_c . The computational power (P) required to

make the MBPC controller implementable in real-time is proportional to the number of generations K that can be evaluated in a computing interval T_c , namely:

$$P \propto K / H \quad (24)$$

Table 3 shows the value of the mean value of variable $\bar{e}(\xi)$ for different values of K and H . Some general design considerations can be drawn. The computing power P required to implement in real-time the algorithm keeps almost constant along the same diagonal of table 3. It is not surprising that, in this ideal case, controllers with the same value of P give comparable performance. Actually, the increase of the prediction error with the increase of the realignment period has no effect; therefore in case of not significant modeling error, given a certain computing power, the choice of the realignment period is not critical.

mean value on 20 exp.		H ($T_c=H \cdot T_s$)				
		1	2	4	8	16
K Gen	1	0.0191	0.0196	0.0201	0.0229	0.0349
	2	0.0187	0.0189	0.0195	0.0203	0.0279
	4	0.0167	0.0180	0.0189	0.0196	0.0223
	8	0.0159	0.0176	0.0185	0.0189	0.0212
	16	0.0148	0.0171	0.0178	0.0180	0.0204
	32	0.0147	0.0166	0.0176	0.0177	0.0193

Table 3. Mean absolute tracking error for $\bar{e}(\xi)$, (Case A)

- *Case B, significant model uncertainty:* To examine the effect of modeling uncertainty, we assumed an inaccuracy in the value of the mass of the pendulum in the prediction model; its value was increased of 30% with respect to the nominal one. Fig. 8 shows the comparison between the system output (solid line) and the output predicted by the model (dashed line) for the Evolutionary MBPC controller characterized by parameters $K=32$ and $H=16$. The two systems are driven by the same optimal input signal calculated online on the basis of the inaccurate prediction model. At the realignment instants the modeling error is zeroed, subsequently it increases due to the system/model mismatch. The corresponding performance of the evolutionary MBPC are reported in table 4. Some general consideration can be drawn. As expected, due to the system/model mismatches, a decrease in the controller performance with respect to the case (A) is observed. Given a certain computing power P , the performance degrades significantly by increasing the realignment period. This is due to the fact that the prediction error increases by enlarging the realignment period. For these reasons, in case of significant modeling error, given a defined value of P , it is preferable to choose the controller characterized by the minimum value of T_c . It is worth of mention that by using the basic Evolutionary MBPC (Onnen, 1997) we are limited by the constraint $T_c=T_s$, namely it is possible to implement only the controllers of the first column ($H=1$) of tables 3 and 4. Clearly, the adoption the proposed

intermittent feedback strategy allows more flexibility in the choice of the parameters of the algorithm to achieve the best performance. In particular, it is not required to compute at least one EA generation in one sampling interval, but this can be computed in two or more sampling intervals thus decreasing the computational load; in fact, as shown by the simulation study, there are many controllers with a K/H ratio less than one that give satisfactory performance.

mean value on 20 exp.		H ($T_c=H \cdot T_s$)				
		1	2	4	8	16
K Gen	1	0.0274	0.0311	0.0332	0.0361	0.0498
	2	0.0267	0.0294	0.0318	0.0329	0.0441
	4	0.0249	0.0268	0.0297	0.0332	0.0394
	8	0.0237	0.0258	0.0278	0.0302	0.0366
	16	0.0231	0.0255	0.0273	0.0297	0.0368
	32	0.0220	0.0253	0.0270	0.0293	0.0340

Table 4. Mean absolute tracking error for $\bar{e}(\xi)$, (Case B)

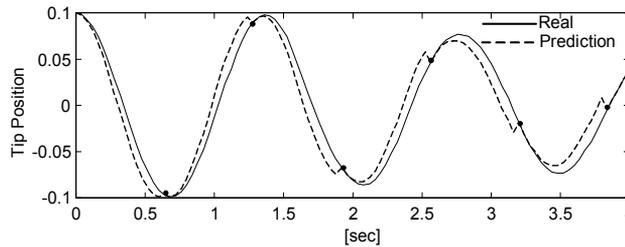


Figure 8. Effect of the system/model realignment in presence of significant modeling error (case $K=32$, $L=16$)

6.3 Repeatability of the control action

The degree of repeatability of the control action of the controllers described in tables 3 and 4 is investigated in this section. The standard deviation (STD) of the variable $\bar{e}(\xi)$ can capture this information; in fact a big STD means that the control action is not reliable (repeatable) and the corresponding controller should not be selected. Table 5 reports the STD of $\bar{e}(\xi)$ in the case of no model uncertainty (case A). In general, given a defined realignment period T_c , the increase of the computational power P reduces the variability in the control action. Acceptable performance can be obtained by employing controllers characterized by the ratio $K/H > 0.5$. Similar considerations are valid also in the case in which the STD is evaluated for a significant model error (case B) and for this reason are not reported.

	$L \quad (T_c=H \cdot T_s)$				
	1	2	4	8	15
1	5.857e-4	9.319e-4	0.0021	0.0039	0.0044
2	4.395e-4	7.736e-4	0.0018	0.0033	0.0042
K 4	4.217e-4	5.759e-4	0.0023	0.0023	0.0015
8	3.802e-4	5.682e-4	9.340e-4	0.0011	0.0017
16	3.568e-4	3.258e-4	4.326e-4	0.0010	0.0020
32	2.165e-4	2.267e-4	2.896e-4	4288e-4	0.0015

Table 5. Standard Deviation of $\bar{e}(\xi)$ (Case A)

6.4 Comparison with conventional optimization methods

The comparison of the Evolutionary MBPC with respect to conventional methods was first carried in (Onnen, 1997), where it was showed the superiority of the EA on the branch-and-bound discrete search algorithm. In this work the intention is to compare the performance provided by the population-based global search provided by an EA with a local gradient-based iterative algorithm. We implemented a basic gradient steepest descent algorithm and used the standard gradient projection method to fulfill the amplitude and rate constraints for the control signal (Kirk,1970); the partial derivatives of the index J with respect to the decision variables were evaluated numerically. Table 6 reports the result of the comparison of the performance provided by the two methods regarding the simulation time, the mean absolute tracking error \bar{e} and the number of simulations required by the algorithm, by varying the number of algorithm cycles K in the case $T_c=T_s$. The Evolutionary MBPC gave remarkably better performance than the gradient-based MBPC regarding the performance \bar{e} ; furthermore, the Evolutionary MBPC requires a minor number of simulations that imply also a minor simulation time. This comparison clearly shows that, in this case, the gradient-based optimization get trapped in local minima, while the EA provides an effective way to prevent the problem.

K	GA ($L_s=32, N=20$)			GRAD ($L_s=32$)		
	N° cycles	sim Time	\bar{e}	N° sim	sim Time	\bar{e}
1	3.79	1.55	20	4.16	1.51	32
2	5.99	0.33	40	6.81	0.66	64
4	9.31	0.33	80	12.08	0.78	128
8	15.99	0.34	160	30.25	0.59	256
16	41.13	0.32	320	50.32	0.54	512
32	65.87	0.35	640	92.81	0.52	1024

Table 6. Comparison between GA and Gradient Optimization

7. Experimental Results

Basing on the results of the previous analysis, we were able to derive the guidelines to implement the improved Evolutionary MBPC for the real-time control of the experimental laboratory system of Fig. 2. The EA was implemented by means of a C procedure and the 4th order Runge-Kutta method was used to perform the time domain integration of the prediction model (6) of the flexible system. As in section 5, the scope of the control system is to damp out the oscillations of the tip of the flexible beam, that starts in the deflected position $\theta = 0.1 \text{ rad}$, $\dot{\theta} = 0$, $\phi = 0$, $\dot{\phi} = 0$. The desired target position is $y_d(k) = \theta_d(k) = 0$. Fig. 9 shows the experimental free response of the tip position that put into evidence the very small damping of the uncontrolled structure. In the same figure it is reported the response obtained by employing a co-located dissipative PD control law of the form

$$\tau(k) = -0.1\phi(k) - 0.5\hat{\phi}(k) \tag{25}$$

where velocity $\hat{\phi}(k)$ has been estimated by means of the following discrete time filter

$$\hat{\phi}(k) = 0.78\hat{\phi}(k-1) + 10[\phi(k) - \phi(k-1)] \tag{26}$$

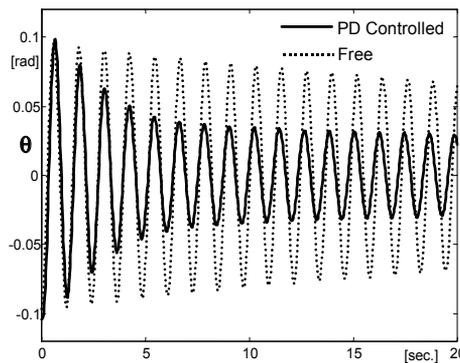


Figure 9. Experimental response

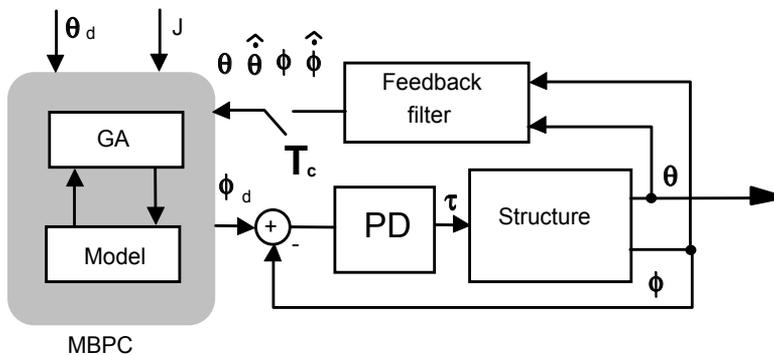


Figure 10. The MBPC scheme for the experimental validation

The conventional PD controller is not able to add a satisfactory damping to the nonlinear system. It is expected that a proper MBPC shaping of the controlled angular position $\phi_i(k)$ of the pendulum could improve the damping capacity of the control system. Fig. 10 shows the block diagram of the implemented MBPC control system.

In order to perform system/model realignment, it is necessary to realign intermittently all the states of the prediction model (6) basing on the measured variables. Because of only positions $\theta(k)$ and $\phi(k)$ can be directly measured by the optical encoders, it was necessary to estimate both the beam and pendulum velocities. The velocity were estimated with a sufficient accuracy by applying single pole approximate derivative filters to the corresponding positions; the discrete time filter for $\hat{\phi}(k)$ is eq (26); $\hat{\theta}(k)$ is estimated by:

$$\hat{\theta}(k) = 0.78\hat{\theta}(k-1) + 10[\theta(k) - \theta(k-1)] \quad (27)$$

Every T_c seconds the vector $[\theta \ \hat{\theta} \ \phi \ \hat{\phi}]$ is passed to the MBPC procedure to perform the realignment. For the reasons explained in section 3, the redefined inputs of the system is the pendulum position $\phi(k)$; therefore a PD controller has been designed to guarantee an accurate tracking of the desired optimal pendulum shaped position $\phi_d(k)$; the PD regulator is:

$$\tau(k) = -5.0(\phi(k) - \phi_{des}(k)) - 0.6\left(\hat{\phi}(k) - \dot{\phi}_{des}(k)\right) \quad (28)$$

The accurate tracking of the desired trajectory $\phi_d(k)$ is essential for the validity of the predictions carried out by exploiting the model (6). Fig. 11 shows the comparison of shaped reference $\phi_d(k)$ and the measured one $\phi(k)$ in a typical experiment. The tracking is satisfactory and the maximum error $|\phi(k) - \phi_d(k)|$ during the transient is 0.11 rad. This error is acceptable for the current experimentation. Note that this PD regulator has a different task respect to regulator (25), employed in the test of Fig. 9; in fact regulator (28) is characterized by higher gains to achieve trajectory tracking, which cause almost a clamping of the pendulum with the beam.

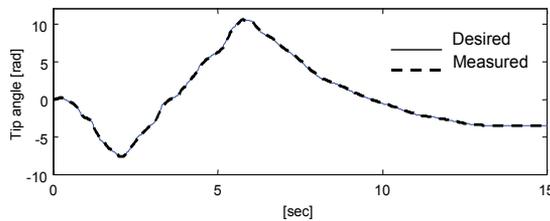


Figure 11. On-line shaped $\phi_d(t)$ and $\phi(t)$ ($T_c = 2T_s$)

7.1 Settings of the experimental evolutionary MBPC

The settings of the MBPC are the same used for the simulations of section 5 and reported in table 2. The decision variables are the sequence of the control input increments $[\Delta u(k+1) \Delta u(k+2) \dots \Delta u(k+N_2)]$. The corresponding input signals $\phi_d(k), \dot{\phi}_d(k), \ddot{\phi}_d(k)$ are obtained by integration of the nominal model equations (6) driven by the sub-optimal control input sequence determined in real-time by the MBPC. The choice of the realignment period T_c is strongly influenced by the available computational power. In this experiment at least two sampling periods T_s are required to compute one generation of the EA when the settings of table 2 are employed, therefore it cannot be implemented with a standard Evolutionary MBPC. On the other hand, the improved algorithm can be easily implemented in real-time by choosing a computing power ratio $K/H \leq 0.5$. An idea of the performance achievable can be deduced by inspecting tables 2, 3 and 4.

7.2 Results

In the experimental phase, it has been evaluated the performance of the MBPC for 4 values of the realignment period T_c ($T_c = HT_s$, $H = [2, 4, 8, 16]$) in the case of a computing power $K/H = 0.5$. Figs. 12 a-d show the measured tip position and the respective value of index \bar{e} for different values of T_c . In all the laboratory experiments a significant improvement of the performance with respect to the co-located PD controller (25) is achieved. In fact, after about 6 seconds the main part of the oscillation energy is almost entirely damped out. In all the experiments the performance does not undergo a significant degradation with the increase of the realignment period, showing that in this case an accurate model of the system has been worked out. The values of the index \bar{e} are in good agreement with the corresponding predicted in table 3 in the case of small modeling error. Anyway, in the case $T_c = 2 T_s$ (Fig. 12a) a superior performance was achieved near the steady state; in this case, the prediction error is minimum and the residual oscillations can be entirely compensated. On the other hand, in the case $T_c = 16 \cdot T_s$ (Fig. 12d) some residual oscillations remain, because the prediction error becomes large due to the long realignment period. To underline the effects of the realignment, in Fig. 13 the error $\theta_d(k) - \theta(k)$ in the case $T_c = 16 \cdot T_s$ is reported. Every 0.64 seconds, thanks to the realignment, the prediction error is zeroed and a fast damping of the oscillations is achieved; near the steady state, the occurrence of high frequency small amplitude oscillations, cannot be recovered effectively. Fig. 14 reports the sequence of the sub optimal control increments applied to the system for the experiment of Fig. 12a. As expected, the adoption of the adaptive mutation range drives to zero the sequence of control increments near the steady state, allowing a very accurate tracking of the desired trajectory. As for the repeatability of the control action no significant difference was observed on the performance in comparable experiments. Repeating 10 times the experiment of Fig. 12a gave a mean of 0.0205 for the $\bar{e}(\zeta)$ index and a standard deviation of 1.112e-3; these are in good accordance with the predicted results of table 5.

The results of the experiments clearly demonstrate that the proposed improved Evolutionary MBPC is able to guarantee an easy real-time implementation of the algorithm giving either excellent performance and a high degree of repeatability of the control action.

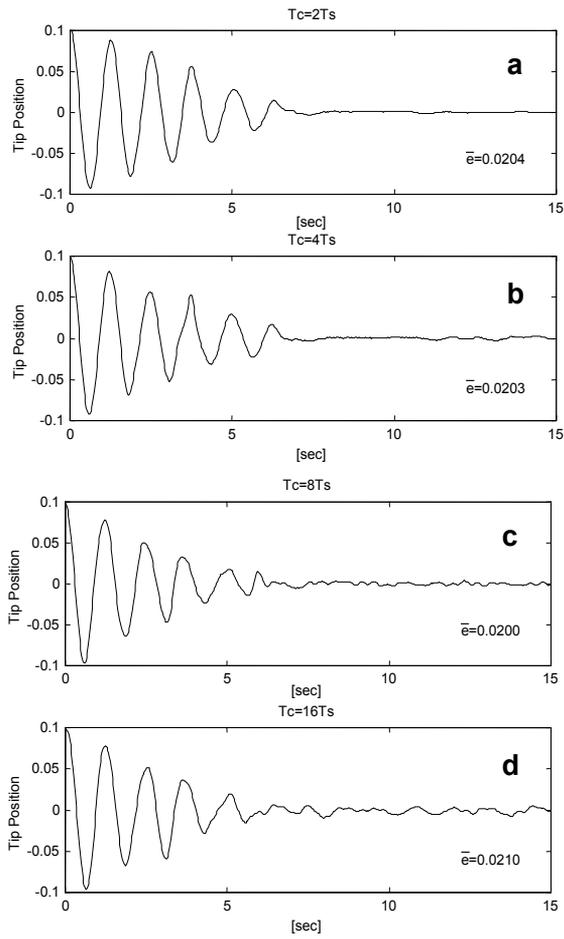


Figure 12. Measured tip position $\theta(t)$ for different values of the system/model realignment time T_c

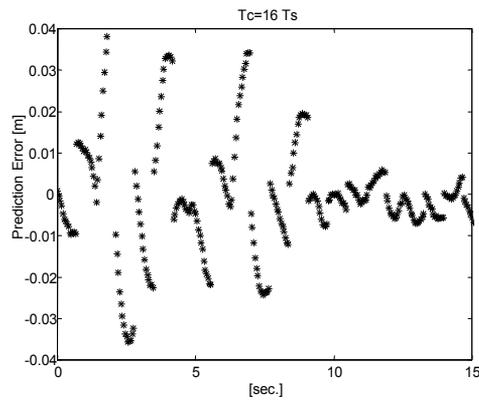


Figure 13. The effect of the system/model realignment on $\theta_d(t) - \theta(t)$

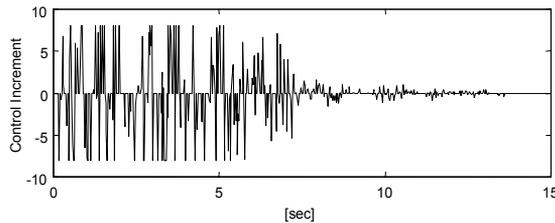


Figure 14. The sequence of input increments in the case $T_c = 2T_s$.

8. Conclusion

This work introduces an improved Evolutionary Algorithm for the real-time Model Based Predictive Control of nonlinear dynamical systems. The main issues involved in the practical real-time implementation of this control scheme have been pointed out and addressed by adapting and extending some known concepts of the conventional Evolutionary MBPC. The advantage of the online adaptation of the dimension of the search space has been pointed out and a new adaptive mutation range operator in function of the actual prediction error has been proposed. The problem related to the computational delay has been faced by inserting an intermittent feedback strategy in the basic Evolutionary MBPC. This extension allows the computation of one generation of the EA in more than one sampling interval, thus decreasing the required computational power for the real-time implementation. The application of the improved EA allows the real-time MBPC of fast dynamics systems by employing a CPU with a limited computing power. The improved algorithm has been experimented with remarkable results for the stabilization of oscillations of a laboratory nonlinear flexible mechanical system. A stochastic analysis showed that improved Evolutionary Algorithm is reliable in the sense that a good repeatability of the control action can be achieved; furthermore, the EA outperforms a conventional iterative gradient-based optimization procedure. Although the potentiality of the improved Evolutionary MBPC have been shown only for a single laboratory experiments, the analysis and design guidelines are general and for this reason can be easily applied to the design of real-time Evolutionary MBPC for a general nonlinear constrained dynamical systems.

9. Predictive Reference Shaping for Constrained Robotic Systems Using Evolutionary Algorithms

Part of the following article has been previously published in: M.L. Fravolini, A. Ficola, M. La Cava. "Predictive Reference Shaping for Constrained Robotic Systems Using Evolutionary Algorithms", Applied Soft Computing, Elsevier Science, in stampa, vol. 3, no. 4, pp.325-341, 2003, ISSN:1568-4946.

The manufacturing of products with a complex geometry demands for efficient industrial robots able to follow complex trajectories with a high precision. For these reasons, tracking a given path in presence of task and physical constraints is a relevant problem that often occurs in industrial robotic motion planning. Because of the nonlinear nature of the robot dynamics, the robotic optimal motion-planning problem cannot be usually solved in closed form; therefore, approximated solutions are computed by means of numerical algorithms.

Many approaches have been proposed concerning the constrained robot motion-planning problem along a pre specified path, taking into account the full nonlinear dynamics of the manipulator. In the well-known technique described in (Bobrow et Al. 1985; Shin, & McKay, 1985), the robot dynamics equations are reduced into a set of second order equations in a path parameter. The original problem is then transformed into finding the curve in the plane of the path parameter and its first time derivative, while the constraints on actuators torque are reduced into bounds on the second time derivative of the path parameter. Although this approach can be easily extended to closed loop robot dynamics, it essentially remains an *offline*-planning algorithm; indeed, in presence of unmodeled dynamics and measurement noise this approach reduces its effectiveness when applied to a real system. Later, to overcome these robustness problems, some *online feedback* path planning schemes have been formulated. Dahl in (Dahl & Nielsen, 1990) proposed an online path following algorithm, in which the time scale of the desired trajectory is modified in real time according to the torque limits. A similar approach has been also proposed by Kumagai (Kumagai et Al., 1996).

Another approach to implement an online constrained robotic motion planning is to employ Model Predictive Control (MPC) strategies (Camacho & Bordons, 1996; Garcia et Al., 1989). A MPC, on the basis of a nominal model of the system, online evaluates a sequence of future input commands minimizing a defined index of performance (tracking error) and taking into account either input or state constraints. The last aspect is particularly important because MPC allows the generation of sophisticated optimal control laws satisfying general multiobjective constrained performance criteria.

Since only for linear systems (minimizing a quadratic cost function) it is possible to derive a closed form solution for MPCs, an important aspect is related to the design of an efficient MPC optimization procedure for the online minimization of an arbitrary cost function, taking into account system nonlinearities and constraints. Indeed, in a general formulation, a constrained non-convex nonlinear optimization problem has to be solved on line, and in case of nonlinear dynamics, the task could be highly computationally demanding; therefore, the online optimization problem is recognized as a main issue in the implementation of MPC (Camacho & Bordons, 1996). Many approaches have been proposed to face the online optimization task in nonlinear MPC. A possible strategy, as proposed in (Mutha et Al., 1997; Ronco et Al., 1999) consists of the linearization of the dynamics at each time step and of the use of standard linear predictive control tools to derive the control policies. Other methods utilize iterative optimization procedures, as gradient based algorithms (Song & Koivo, 1999) or discrete search techniques as Dynamic Programming (Luus, 1990) and Branch and Bound (Roubos et Al., 1999) methods. The main advantages of a search algorithm is that it can be applied to general nonlinear dynamics and that the structure of the objective function is not restricted to be quadratic as in most of the other approaches. A limitation of these methods is the fast increase of the algorithm complexity with the number of decision variables and grid points. Recently, another class of search techniques, called Evolutionary Algorithms (EAs) (Foegel, 1999; Goldberg, 1989) showed to be very effective in *offline* robot path planning problems (Rana, 1996); in the last years, thanks to the great advancements in computing technology, some authors have also proposed the application of EAs for *on-line* performance optimization problems (Lennon & Passino, 1999; Liaw & Huang, 1998; Linkens & Nyongesa, 1995; Martinez et Al., 1998; Onnen et Al., 1997; Porter & Passino, 1998).

Recently the authors, in (Fravolini et Al., 2000) have applied an EA based optimization procedure for the online reference shaping of flexible mechanical systems. The practical *real-time* applicability of the proposed approach was successively tested with the experimental study reported in (Fravolini et Al., 1999). In this work the approach is extended to the case of robotic motion with constraints either on input and state variables. Two significant simulation examples are reported to show the usefulness of the online reference shaping method; some considerations concerning the online computational load are also discussed. The paper is organized as follows. Section 10 introduces the constrained predictive control problem and its formalization. Section 11 introduces the EA paradigm, while section 12 describes in details either the online EA based optimization procedure and the specialized EA operators required for MPC. In section 13 the proposed method is applied to two benchmark systems; in section 14 a comparison with a gradient-based algorithm is discussed. Finally, the conclusions are reported in section 15.

10. The Robotic Constrained Predictive Control Method

In the general formulation it is assumed that an inner loop feedback controller has already been designed to ensure stability and tracking performance to the robotic system, as shown is figure 15. In case of fast reference signals $r(k)$, the violation of input and state constraints could occur; to overcome this problem the authors, in (Fravolini et Al., 2000) proposed to add to the existing feedback control loop an online predictive reference shaper. The predictive reference shaper is a nonlinear device, that modifies in real-time the desired reference signal $r(k)$ on the basis of a prediction model and of the current feedback measures $y(k)$. The scope is that the shaped reference signal $r_s(k)$ allows a more accurate track of the reference signal without constraints violation. Usually, these requirements are quantified by an index of cost J and a numerical procedure is employed to online minimize this function with regard to a set of decision variables. Typically, the decision variables are obtained by a piecewise constant parameterization of the shaped reference $r_s(k)$.

As for predictive controllers strategies, the reference shaper evolves according to a receding horizon strategy: the planned sequence is applied until new feedback measures are available; then a new sequence is calculated that replaces the previous one. The receding horizon approach provides the desired robustness against both model and measurement disturbance.

Note 1: The trajectory shaper of figure 15 could also be employed without the inner feedback control loop. In this case, the shaper acts as the only feedback controller and it directly generates the control signals; in this case $u(k) \equiv r_s(k)$.

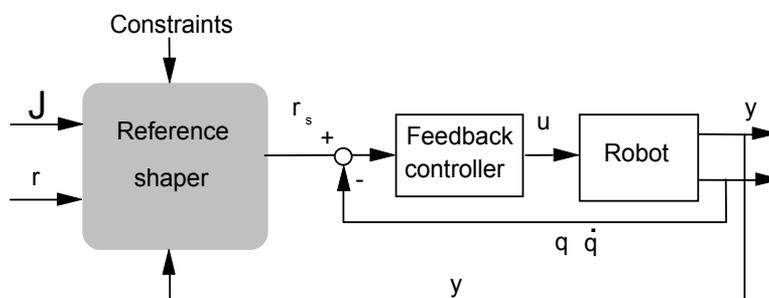


Figure 15. Online reference shaper

10.1 Problem Formulation

The equation of motion of a constrained robot dynamics can be formally expressed as follows:

$$\begin{cases} M(q)\ddot{q} + C(q, \dot{q}) + G(q) = Qu \\ u = u(q, \dot{q}, \xi, r_s) \quad y = F \cdot [q^T \quad \dot{q}^T]^T \\ h = h(q, \dot{q}, \xi, r_s) \\ q(0) = q_o \quad \dot{q}(0) = \dot{q}_o \quad \xi(0) = \xi_o \end{cases} \quad (1)$$

where $q \in R^m$ is the vector of robot positions, \dot{q} is the vector of robot velocities, ξ collects the states either of the controller or of the actuation system, M is the inertia matrix, C is the Coriolis vector, G the gravity vector, Q is the input selection matrix, $u \in R^m$ is the control vector, $y \in R^n$ is the output performance vector, F is the output selection matrix, $r_s \in R^n$ is the shaped reference to be tracked by q . The vector $[q_o, \dot{q}_o, \xi_o]$ represents the initial condition of the feedback-controlled system; $h \in R^p$ is the vector of the p constraints that must satisfy the relation:

$$h(q, \dot{q}, \xi, r_s, t) \in H \quad t > 0 \quad (2)$$

where H is the set where all the constraints (2) are fulfilled.

The aim of the reference shaper is to online compute the shaped reference $r_s(k)$ in order to fulfill all the constraints (2) while minimizing a defined performance measure J that is function of tracking error of the performance variables y . Since the online optimization procedure requires a finite computing time T_c (optimization time) to compute a solution, the proposed device is discrete time with sampling instants $k_c T_c$. The inner feedback controller is allowed to work at a faster rate kT_s (where $T_c = a \cdot T_s$ and $a > 1$ is an integer, therefore $k_c = a \cdot k$). The optimal sequence $r_s^*(k_c)$ is determined as the result of an optimization problem during each computing interval T_c . The cost index J , which was employed, is:

$$J(k_c) = \sum_{i=1}^n \alpha_i \sum_{j=1}^{N_{Yi}} |\hat{y}_i(k_c + j | k_c) - r_i(k_c + j)| + \sum_{i=1}^m \beta_i \sum_{j=1}^{N_{Ui}} |\Delta u_i(k_c + j - 1)| + J_1(k_c) \quad (3)$$

The first term of (3) quantifies the absolute predicted tracking error between the desired output signal $r_i(k_c + j)$ and the predicted future output $\hat{y}_i(k_c + j | k_c)$, estimated on the basis of the robot model and the feedback measures available at instant k_c ; this error is evaluated over a defined prediction horizon of N_{Yi} samples. The second term of (4) is used to weight the control effort that is quantified by the sequence of the input increments $\Delta u_i(k) = u_i(k) - u_i(k-1)$ (evaluated over the control horizon windows of N_{Ui} samples). The coefficients α_i and β_i are free weighting parameters. The term $J_1(k_c)$ in (5) is a further cost function, which can be used to take into account either task or physical constraints. In this work the following constraints on control and state variables have been considered:

$$U_i^- < u_i(k) < U_i^+ \quad \Delta U_i^- < \Delta u_i(k) < \Delta U_i^+ \quad i = 1, \dots, n \quad (6)$$

$$X_i^- < x_i(k) < X_i^+ \quad \Delta X_i^- < \Delta x_i(k) < \Delta X_i^+ \quad i = 1, \dots, n \quad (7)$$

Constraints (6) take into account possible saturations in the amplitude and rate of the actuation system, while constraints (7) prevent the robot to work in undesirable regions of the state space. The optimization problem to be solved during each sampling interval T_c is formalized as follows:

$$\underset{R_{si}(k_c)}{\text{mim}} J(k_c) \quad (8)$$

taking into account the fulfillment constraints (6) and (7). The optimization variables of the problem are the elements of the sequences $R_{si}(k_c)$ evaluated within the control horizon:

$$R_{si}(k_c) = [\Delta r_{si}(k_c), \Delta r_{si}(k_c + 1), \dots, \Delta r_{si}(k_c + N_{ui} - 1)] \quad i = 1, \dots, n \quad (9)$$

11. Evolutionary Algorithms

The Evolutionary Algorithms are multi point search procedures that reflect the principle of evolution, natural selection and genetics of biological systems (Foegel; 1994; Goldberg, 1989). An EA explores the search space by employing stochastic rules; these are designed to quickly direct the search toward the most promising regions. It has been shown that the EAs provide a powerful tool that can be used in optimization and classification applications. The EAs work with a population of points called chromosomes; a chromosome represents a potential solution to the problem and comprises a string of numerical and logical variables, representing the decision variables of the optimization problem. The EA paradigm does not require auxiliary information, such as the gradient of the cost function, and can easily handle constraints; for these reasons EAs have been applied to a wide class of problems, especially those difficult for hill-climbing methods.

An EA maintains a population of chromosomes and uses the genetic operators of "selection" (it represents the biological survival of the fittest ones and is quantified by a fitness measure that represents the objective function), "crossover" (which represents mating), and "mutation" (which represents the random introduction of new genetic material), with the aim of emulating the evolution of the species. After some generations, due to the evolutionary driving force, the EA produces a population of high quality solutions for the optimization problem.

The implementation of a basic EA can be summarized by the following sequence of standard operations:

- a. (Random) Initialization of a population of N solutions
- b. Calculation of the fitness function for each solution
- c. Selection of the best solutions for reproduction
- d. Application of crossover and mutation to the selected solutions
- e. Creation of the new population
- f. Loop to point b) until a defined stop criterion is met

12. The Online Optimization Procedure for MPC

In order to implement the predictive shaper described in the previous section, it is necessary to define a suitable online optimization procedure. In this section it is described the MPC

algorithms based on an EA; the resulting control scheme is reported in figure 16. The flow diagram of the online optimization procedure implemented by the Evolutionary reference shaper is reported in figure 17.

Note 2: In this work, to keep notation simple, the prediction (N_{Yi}) and control (N_{Ui}) horizons are constrained to have the same length for each output and each input variable ($N_Y = N_U$).

Note 3: Because the feedback controller sampling interval T_s is often different from the optimization time T_c ($T_c = a \cdot T_s$, often $T_c \gg T_s$), during a period T_c it is required to perform predictions with a prediction horizon N_Y longer at least $2T_c$ ($2a$ samples, $T_c = 2a \cdot T_s$). More precisely, during the current computation interval $[k_c, k_c+1]$ the first a values of the optimal sequences $R_{si}^*(k_c)$ that will be applied to the real plant are fixed and coincide with the optimal values computed in the previous computational interval $[k_c-1, k_c]$; the successive $N_U - a$ values are the actual optimization variables that will be applied in the successive computation interval $[k_c+1, k_c+2]$.

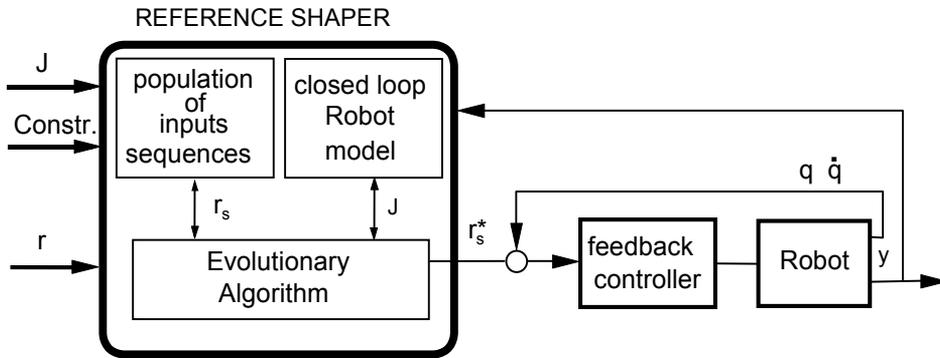


Figure 16. The proposed evolutionary reference shaper

12.1 Specialized Evolutionary Operators for MPC

The application of EA within a MPC requires the definition of evolutionary operators expressly designed for real time control. These operators were introduced in previous works (Fravolini et Al, 1999; Fravolini et Al. 2000) and tested by mean of extensive simulation experiments. Some of these operators are similar to those reported in (Goggos; 1996; Grosman & Lewin, 2002; Martinez et Al., 1998; Onnen et Al., 1997) and constitute a solid, tested, and accepted base for evolutionary MBPC.

In this paragraph the main EA operators expressly specialized for online MPC are defined.

- *Fitness function:* The objective function to be online minimized (with rate T_c) is the index $J(k_c)$ in (3). The fitness function is defined as:

$$f = 1/J \quad (10)$$

- *Decision variables:* The decision variables are the elements of the sequences $R_{si}(k_c)$ (9).
- *Chromosome structure and coding:* A chromosome is generated by the juxtaposition of the coded sequence of the increments of the shaped reference $R_{si}(k_c)$. With regard to the codification of the decision variables, some alternatives are possible. Binary or decimal codification are not particularly suited in online applications, since they require time consuming coding and decoding routines. Real coded variables, although do not require any codification, have the drawback that the implementation of evolutionary

operators for real numbers is significantly slower than in the case of integers variables. Therefore, the best choice is an integer codification of the decision variables, which can guarantee a good accuracy of the discretization while operating on integer numbers. A coded decision variable x_{ij} can assume an integer value in the range $0, \dots, L_0, \dots, L_i$, where L_i represents the quantization accuracy. This set is uniformly mapped in the bounded interval $\Delta R_i^- \leq \Delta r_{si} \leq \Delta R_i^+$. L_0 represents the integer corresponding to $\Delta r_{si} = 0$. The l -th chromosome in the population at t -th generation during the computing interval $[k_c, k_c+1]$ is a string of integer numbers:

$$X_l^t = [x_{11}, x_{12}, \dots, x_{1, N_{U1}} \mid x_{21}, x_{22}, \dots, x_{2, N_{U2}} \mid \dots, x_{m1}, x_{m2}, \dots, x_{m, N_{Um}}] \quad (11)$$

the actual value of the decision variables $\Delta r_{si}(k_c+j)$ are obtained by applying the following decoding rule:

$$\Delta r_{si}(k+j) = \Delta R_i^- + \Delta R_i \cdot x_{i,j} \quad j=1, \dots, N_{Ui} \quad i=1, \dots, n \quad (12)$$

where $\Delta R_i = \left| \Delta R_i^+ - \Delta R_i^- \right| / L_i$ is the control increment resolution for the i -th input. The sequences of shaped references applied in the prediction window result:

$$r_{si}(k_c+j) = r_{si}(k_c-1+j) + \Delta r_{si}(k_c+j) \quad j=1, \dots, N_{Ui} \quad i=1, \dots, n \quad (13)$$

- *Selection and Reproduction mechanisms (during a computing interval T_c)* Selection and reproduction mechanisms act at two different levels during the on-line optimization. The lower level action concerns the optimization of the fitness function f during T_c . Because a limited computation time is available, it is essential that the good solutions founded in the previous generations are not lost, but are used as "hot starters" for the next generation. For this reason a *steady state* reproduction mechanism is employed, namely the best S chromosomes in the current generation are passed unchanged in the next one; this ensures a not decreasing fitness function for the best population individual. The remaining part of the population is originated on the basis of a *rank selection* mechanism; given a population of size N , the best ranked D individuals constitute a mating pool. Within the mating pool two random parents are selected and two child solutions are created applying crossover and mutation operators; this operation lasts until the new population is entirely replaced. This approach is similar to the algorithm described in (Yao & Sethares, 1994).
- *Receding Horizon Heredity (between two successive computational intervals)*: The second level of selection mechanism implements the receding horizon strategy. The best chromosomes computed during the current T_c are used as starting solutions for the next optimization. At the beginning of the next computational interval, because the prediction and control horizons are shifted in the future, the values of the best S' chromosomes are shifted back of a locations ($T_c = a \cdot T_s$). In this way the first a values are lost and their positions are replaced by the successive a . The values in the last positions (from $a+1$ to N_{Ui}) are simply filled by keeping constant the value of the a -th variable. The shifted S' chromosomes represent the "hot starters" for the optimization in the next computational interval; the remaining $N-S'$ chromosomes are randomly generated. The

application of hereditary information is of great relevance, because a significant improvement in the convergence speed of the algorithm has been observed.

- *Crossover*: during an optimization interval T_c uniform crossover has been implemented. Given two selected chromosomes X_a^t and X_b^t in the current generation t , two corresponding variables are exchanged with a probability p_c , namely the crossed elements in the successive generation result:

$$x_{a,(i,j)}^{t+1} = x_{b,(i,j)}^t \text{ and } x_{b,(i,j)}^{t+1} = x_{a,(i,j)}^t \quad (14)$$

- *Mutation*: random mutation are applied with probability p_m to each variable of a selected chromosome X_a^t , in the current generation according to the formula:

$$x_{a,(i,j)}^{t+1} = x_{a,(i,j)}^t + \text{rand}(\bar{\Delta}) \quad (15)$$

where $\text{rand}(\bar{\Delta})$ is a random integer in the range: $[-\Delta_{xi}, \dots, 0, \dots, \Delta_{xi}]$ and Δ_{xi} is the maximum mutation amplitude.

- *Constraints*: One of the main advantages of MPC is the possibility of taking into account of constraints during the online optimization. Different kinds of constraints have been considered:

i) *Constraints on the shaped reference*: In order to generate references that could be accurately tracked by means of the available actuation system, it can be required to constrain either the maximum/minimum value of the shaped signals or and their rate of variation. For this reason, if, during the optimization a decision variable $x_{i,j}$ violates its maximum or minimum allowed value in the range $0, \dots, L_i$, the following threshold is applied:

$$\begin{cases} \text{if } x_{i,j}^t > L_i \text{ then } x_{i,j}^t = L_i \\ \text{if } x_{i,j}^t < 0 \text{ then } x_{i,j}^t = 0 \end{cases} \quad i = 1, \dots, n \quad (16)$$

The application of (16) automatically guarantees that $\Delta R_i^- \leq \Delta r_{si}(k) \leq \Delta R_i^+$. In a similar fashion it is possible to take into account of the constraint on the amplitudes $R_i^- \leq r_{si}(k) < R_i^+$ by applying the thresholds:

$$\begin{cases} \text{if } r_{si}(k) + \Delta r_{si}(k) > R_i^+ \text{ then } x_{i,k}^t = L_0 + \text{int} \left(\frac{R_i^+ - r_{si}(k)}{\Delta_{Ri}} \right) \\ \text{if } r_{si}(k) + \Delta r_{si}(k) < R_i^- \text{ then } x_{i,k}^t = L_0 + \text{int} \left(\frac{R_i^- - r_{si}(k)}{\Delta_{Ri}} \right) \end{cases} \quad i = 1, \dots, n \quad (17)$$

Thresholds (16) and (17) ensure the desired behavior of $r_{si}(k)$.

ii) *Constraints on control signals*: In the case the inner control loop is not present, then $u(k) \equiv r_s(k)$ (see Note1) therefore the constraints are automatically guaranteed by thresholds

(16)-(17). In the case the inner loop is present, constraints (18) are implicitly taken into account in the optimization procedure by inserting in the prediction model an amplitude and a rate saturation on the command signals generated by the inner controller. These thresholds are implemented by:

$$\begin{cases} \text{if } u_i(k) > U_i^+ & \text{then } u_i(k) = U_i^+ \\ \text{if } u_i(k) < U_i^- & \text{then } u_i(k) = U_i^- \end{cases} \quad (19)$$

and:

$$\begin{cases} \text{if } \Delta u_i(k) > \Delta U_i^+ & \text{then } \Delta u_i(k) = \Delta U_i^+ \\ \text{if } \Delta u_i(k) < \Delta U_i^- & \text{then } \Delta u_i(k) = \Delta U_i^- \end{cases} \quad (20)$$

Generally, in the design phase of the inner feedback controllers, it is difficult to take into account the effects of the possible amplitude and rate limit on the inputs; on the other hand constraints (19) and (20) are easily introduced in the MPC approach. The resulting shaped references are thus determined by taking explicitly into account the physical limitations of the actuation systems.

iii) Constraints on state variables: These constraints are taken into account by exploiting the penalty function strategy; namely a positive penalty term J_1 , proportional to the constraint violation, is added to the cost function $J(k_c)$ in (21). Let the set H defining the p constraints in (2) be defined as:

$$H = \{h_i \in R^p : h_i(q, \dot{q}, x, r_s) \leq 0, i = 1, \dots, p\} \quad (22)$$

then, the penalty function taking into account the violation of constraints along the whole prediction horizon has been defined as:

$$J_1(k) = \sum_{i=1}^p \gamma_i \sum_{j=1}^{N_y} \max(h_i(k+j), 0) \geq 0 \quad (23)$$

When all the constraints are fulfilled, J_1 is equal to zero, otherwise it is proportional to the integral of the violations. By increasing the values of the weights γ_i , it is possible to enforce the algorithm toward solutions that fulfill all the constraints.

- *Choice of Computing time T_c :* It should be chosen as a compromise between the two concurrent factors:

1) The enlargement of the computing time T_c allows to refine the degree of optimality of the best solutions by increasing the EA generation number, gen , that can be evaluated within an optimization period.

2) A large T_c causes the increase of the delay in the system. Excessive delays cannot be acceptable for fast dynamics systems.

Obviously, the computational time is also influenced by the computing power of the processor employed.

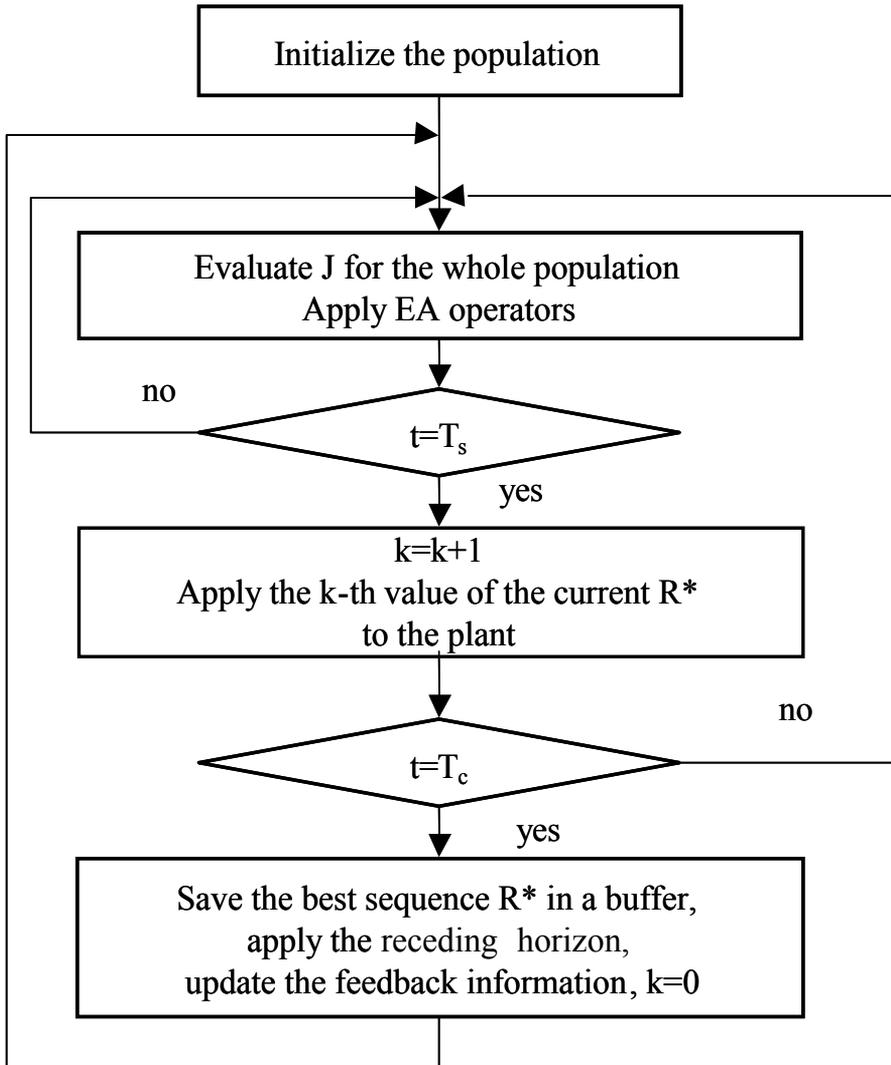


Figure 17. Flow diagram of the evolutionary reference shaper

13 Applications

In this section the proposed evolutionary shaper is applied to two significant robotic control problems.

13.1 Coupled Flexible Beam/Pendulum (Ex. 1)

In this example the performance of the trajectory shaper has been tested on a nonlinear flexible mechanical system. The system is composed of a flexible beam clamped at one side, with a controlled pendulum hinged on the other; the motion occurs in the horizontal plane (figure 18a). The stabilization of this system represents an excellent benchmark because of the fast dynamics and the presence of oscillatory and scarcely damped modes. The scope of

the control law is to add damping to the system by properly driving the pendulum, when the beam starts in a deflected position. The beam is modeled as a first order mass-spring-damper element (figure 18b). The matrices of the system according to model (1) are:

$$M = \begin{bmatrix} (M + m)L^2 + ml^2 - 2mL \cos \phi & ml^2 - mL \cos \phi \\ ml^2 - mL \cos \phi & ml^2 \end{bmatrix}$$

$$C = \begin{bmatrix} 2mL\dot{\phi}\dot{\theta} \sin \phi + mL\dot{\phi}^2 \sin \phi + K_1\theta + C_1\dot{\theta} \\ -mL\dot{\theta}^2 \sin \phi + C_2\dot{\phi} \end{bmatrix} \quad (24)$$

$$Q = [0 \ u]^T \quad q = [\theta \ \phi]^T$$

The parameters of the model (25) are reported in table 7. The inner loop is controlled by a PD law; this controller was expressly tuned to increase the damping effect induced by the controlled pendulum on the beam oscillations. In figure 19a it is reported the deflection of the tip position of the beam in the case of free and of PD-controlled response, while in figure 19b it is reported the corresponding PD control signal $u(k)$. Although a significant reduction of the oscillations is observed, the PD controller is not able to add a large damping to the system.

To improve the performance of the PD controlled system a reference MPC shaper is added to the inner loop according to general scheme of figure 15. The decision variables are the elements of the sequence $[\Delta r_s(k_c+1), \Delta r_s(k_c+2), \dots, \Delta r_s(k_c+N_U-1)]$ of the shaped $r_s(k)$ reference for the feedback PD controller. The horizon length was set to $N_U=N_Y=32$; this implies a control horizon of $N_U T_s = 1.28$ s ($T_s=0.04$ s); the employment of shorter horizons has generated unsatisfactory responses. The other parameters of the evolutionary shaper are reported in table 8. Since the scope is to damp out the oscillation of the tip of the beam, the objective function (25) was particularized as follows:

$$J(k_c) = \sum_{j=0}^{N_U} |\theta(k_c + j) - y_d(k_c + j)| = \sum_{j=0}^{N_U} |\theta(k_c + j) - 0| \quad (26)$$

where the angle $\theta(k)$ is chosen as performance variable ($\theta(k)=y(k)$) (it was assumed that for small deflections the tip position is $L\ddot{\theta}(k)$). Furthermore, a saturation block ($-20 < u(k) < 20$) was added at the output of the PD controller either in the "real" system and in the prediction model to take into account the saturation of the actuation system. Some simulations were performed with the aim of comparing the performance of the reference shaper by varying the computational power required for real-time computation. This aspect has been emulated by fixing the number of generations ($gen=10$) evaluated during the interval T_c and by varying its duration; indeed, the longer is T_c , the less is the computing power required. The computing times $T_c=[1, 2, 4, 8, 16] \cdot T_s$ have been evaluated; the test results are summarized by the performance indexes reported in table 9. The reported indexes were evaluated over a period of 15 s. In column 2 the percent performance increase (evaluated by means of the mean absolute tracking error) with regard to the PD controller case is reported; the shaper produced a significant reduction of about 30 % in the case of

$T_c=[1, 2, 4] \cdot T_s$. Conversely, for $T_c=16 \cdot T_s$ the shaper has an evident negative effect (-56%), because in this case, the number of generations evaluated in a sampling time T_s ($gen/T_s=0.625$, see column 7) is too small to reach a satisfactory solution. In the successive columns of table 9 the mean, maximum and minimum value of the control signal are reported. In column 6 it is reported the ratio (t_s/t_r) between the simulation time t_s employed to run the simulation and the real-time t_r . All the described simulations were carried out by employing a PentiumII 200 MHz processor; the code was written in C, and the dynamics were simulated with a fourth order Runge-Kutta algorithm. In more details, for $T_c \geq 2 \cdot T_s$, the normalized simulation time is less than one; this implies that the proposed procedure could be employed on the physical system with the current processor as real-time controller. In figure 20a it is shown the tip position when the reference shaper is applied when $T_c=2 \cdot T_s$. A significant increase in the oscillation damping is observed with respect to the PD law. In figure 20b the corresponding control signal is shown; it should be noted that, although the active constraint on the command amplitude, the performance are not degraded. In figure 21a the online shaped reference $r_s(k)$ is shown, while in figure 21b it is reported the motor position $\phi(k)$. Overall, the results clearly show that the evolutionary shaper was able to give a substantial improvement to the performance of the existing PD feedback controller.

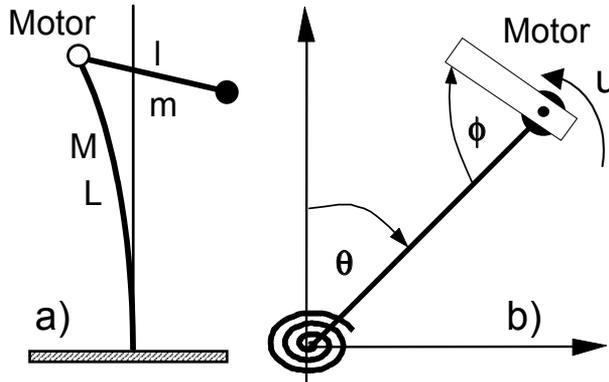


Figure 18. The structure (a), and the model (b)

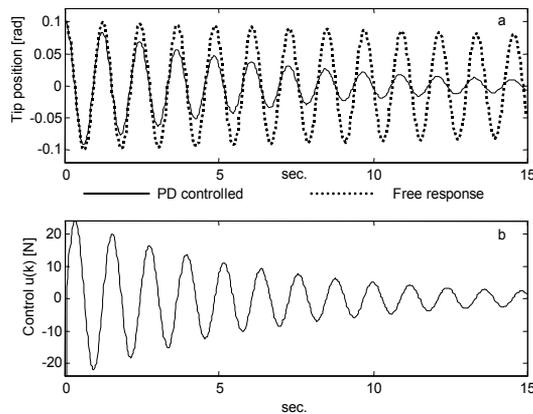


Figure 19. Free and PD controlled response (a), PD control effort (b)

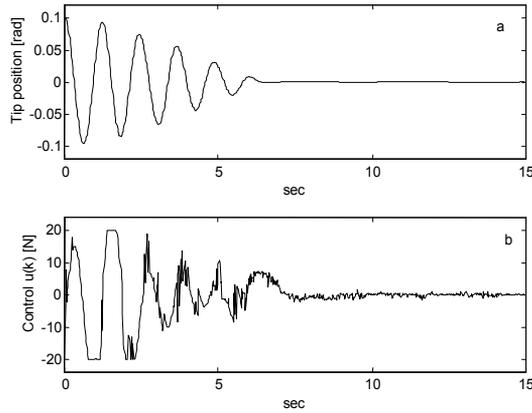


Figure 20. The response with the reference shaper PD+MPC ($T_c = 2 \cdot T_s$)

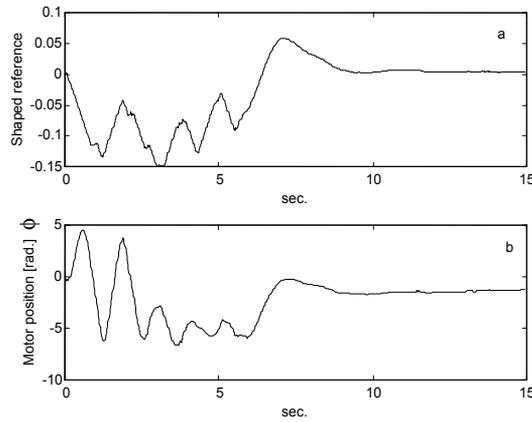


Figure 21. Shaped reference r_s (a) and motor position (b)

M	0.689 kg	Mass of the beam
m	0.070 kg	Mass of the pendulum
L	1.000 m	Length of the beam
l	0.086 m	Length of the pendulum
K_1	25.3 Nm/rad	Elastic coeff. of the beam
C_1	0.008 Nm/s rad	Damping coeff. of the beam
C_2	0.050 Nm/s rad	Damping coeff. of the pendulum

Table 7. Parameters of the model (Example 1)

$N = 20$	$D = 6$	$S = 2$	$S' = 4$
$p_m = 0.1$	$p_c = 0.8$	$T_s = 0.04$	$L = 4000$
$N_U = 32$	$N_Y = 32$	$\Delta_x = 1000$	$R^+ = 0.4$
$R^- = -0.4$	$\Delta R^+ = 0.01$	$\Delta R^- = -0.01$	$gen = 10$

Table 8. Parameters of the evolutionary shaper (example 1)

	T_c	$ e(k) _{mean}$	$ u(k) _{mean}$	U_{max}	U_{min}	t_s/t_r	Gen/T_s
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Free	-	0.0569	-	-	-	-	-
PD	T_s	0.0243 +0%	5.87	21.8	-20.1	-	-
PD+MPC	T_s	0.0164 +32%	4.05	20	-20	1.19	10
PD+MPC	$2 T_s$	0.0167 +31%	4.11	20	-20	0.73	5
PD+MPC	$4 T_s$	0.0193 +20%	5.37	20	-20	0.33	2.5
PD+MPC	$8 T_s$	0.0231 +5%	6.02	20	-20	0.18	1.25
PD+MPC	$16 T_s$	0.0380 -56%	9.27	20	-20	0.11	0.625

Table 9. Performance of PD and PD+MPC (example 1). Col.1 Computing time, Col.2 Mean tracking error, Col.3 mean control effort, Col.4 Maximum control, Col.5 Minimum control, Col.6 Normalized simulation time, Col.7 Generation evaluated in a sampling interval

13.2 Two Links Planar Robot (Ex. 2)

In this study a well-known benchmark system in robotic optimal control was considered (Bobrow et Al., 1985): the two-links planar robot shown in figure 22. The matrices of the model according to (1) are:

$$\begin{aligned}
 M &= \begin{bmatrix} \beta_5 + 2\beta_6 c \phi + \beta_7 & \beta_6 c \phi + \beta_7 \\ \beta_6 c \phi + \beta_7 & \beta_7 \end{bmatrix} \\
 C &= \begin{bmatrix} -\beta_6 \dot{\psi}(2\dot{\theta} + \dot{\phi}) \text{sen}(\phi) \\ -\beta_6 \dot{\phi} \dot{\theta}^2 \text{sen}(\phi) \end{bmatrix} \\
 Q &= \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}
 \end{aligned} \tag{27}$$

where: $\beta_4 = I_2 + m_3 l_2^2$, $\beta_5 = I_1 + l_2^2(m_2 + m_3)$, $\beta_6 = l_1(a_2 m_2 + l_2 m_3)$, $\beta_7 = I_3 + \beta_4$

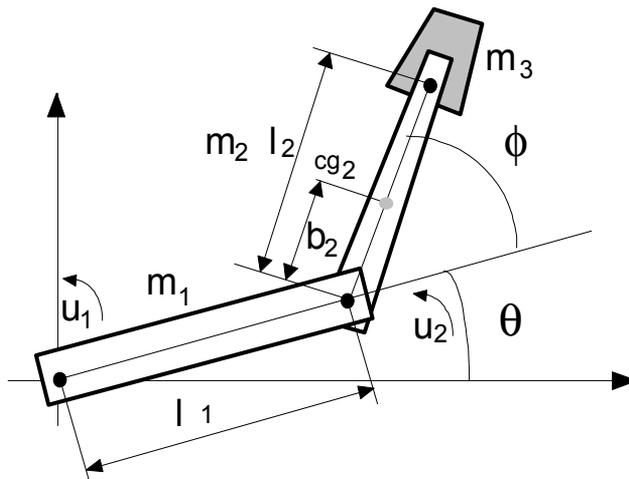


Figure 22. The model of the two-links planar robot

The model parameters are reported in table 10. In this example it has been assumed that the reference shaper directly generates the control torques and for this reason no inner control loop was employed. The desired trajectory to be tracked by the end effector (tip position) is the rectangular trajectory defined (in Cartesian coordinates) by:

$$\begin{cases} x_d = +0.21 \cdot (2 - t) & y_d = 0 & 0 \leq t < 4 \\ x_d = -0.42 & y_d = -10.5 \cdot (t - 4) & 4 \leq t < 8 \\ x_d = -0.21 \cdot (10 - t) & y_d = 0.42 & 8 \leq t < 12 \\ x_d = +0.42 & y_d = 10.5 \cdot (t - 16) & 12 \leq t < 16 \end{cases} \quad (28)$$

Length of the first link (l_1)	0.4 m
Length of the second link (l_2)	0.25 m
Length b_2	0.125 m
Inertia of the 1th. link about its C.G. (I_1)	1.6 m ² ·kg
Inertia of the 2th. link about its C.G. (I_2)	0.43 m ² ·kg
Inertia of the hand w.r. the hand (I_3)	0.01 m ² ·kg
Mass of the second link (m_2)	15 kg
Mass of the load (m_1)	6 kg

Table 10. Parameters of the model (Example 2)

In the simulation study, the first joint was fixed at the origin of the reference frame. In figure 23 the desired trajectory (28) and the corresponding velocities are shown. It has to be noted that, due to the discontinuity of the velocities, it is not possible to track the reference trajectory accurately near the discontinuity; anyway, thanks to the predictive action this effect can be reduced also in presence of constraints. The purpose of the MPC shaper is to achieve a small tracking error while fulfilling the following constraints on inputs:

$$-10 \leq u_1(k) \leq 10 \quad -3 \leq \Delta u_1(k) \leq 3 \quad (29)$$

$$-2 \leq u_2(k) \leq 2 \quad -0.3 \leq \Delta u_2(k) \leq 0.3 \quad (30)$$

and the constraints on joint velocities:

$$-0.9 \leq \dot{\theta}(k) \leq 0.9 \quad -0.9 \leq \dot{\phi}(k) \leq 0.9 \quad (31)$$

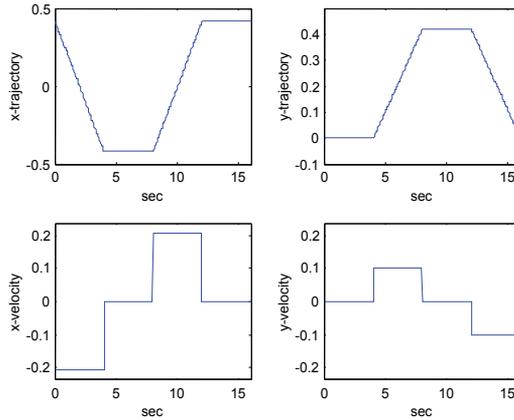


Figure 23. The cartesian trajectory and velocity

All the values are expressed in MKS. The decision variables of the trajectory shaper are the elements of the sequences of the torques that have to be applied at joints, namely:

$$R_s(k_c) = \begin{bmatrix} \Delta u_1(k_c + 1), \Delta u_1(k_c + 2), \dots, \Delta u_1(k_c + N_{u1}) \\ \Delta u_2(k_c + 1), \Delta u_2(k_c + 2), \dots, \Delta u_2(k_c + N_{u2}) \end{bmatrix} \quad (32)$$

The following objective function have been employed:

$$J(k_c) = \sum_{j=1}^{N_U} |x(k_c + j) - x_d(k_c + j)| + |y(k_c + j) - y_d(k_c + j)| + J_1(k_c) \quad (33)$$

where the first term represents the absolute tip position tracking error, while the penalty term J_1 is employed to take into account the constraints on joint velocities and it is defined, according to the penalty approach (34), as:

$$J_1(k) = 5 \left[\sum_{j=1}^{N_U} \max(|\dot{\theta}(k + j)| - 0.9, 0) + \sum_{j=1}^{N_U} \max(|\dot{\phi}(k + j)| - 0.9, 0) \right] \quad (35)$$

Note that in this example, since the trajectory shaper directly generates the command signals ($r_s(k) \equiv u(k)$), it is not required to explicitly insert the thresholds (36) and (37) in the prediction model, because thresholds (38) and (39) directly act on the control signals $[u_1, u_2]$. Some simulations were performed, in the case $T_c = 2T_s$, to determine the appropriate values for the shaper parameters, that are reported in table 11. The performance of the shaper has been detailing tested in three different contexts.

$N = 20$	$D = 6$	$S = 2$	$S = 4$
$p_m = 0.1$	$p_c = 0.8$	gen = 10	$T_s = 0.02$
$L = 4000$	$\Delta_x = 1000$	$N_{u1} = 12$	$N_{u2} = 12$
$\Delta U_1^- = -3$	$\Delta U_1^+ = 3$	$U_1^- = -10$	$U_1^+ = 10$
$\Delta U_1^- = -0.3$	$\Delta U_2^+ = 0.3$	$U_2^- = -2$	$U_1^+ = 2$

Table 11. Parameters of the evolutionary shaper (example 2)

Experiment 1: In this experiment no constraints were imposed either on inputs or on states; the scope was just to obtain a very accurate tracking and therefore the constraints (40)(41) were disabled. In figure 24a it is reported the trajectory of the tip position, while in figure 24b the corresponding absolute tracking error is shown. Although a very good tracking have been achieved, the constraints on inputs, joint velocities and the corresponding rates are violated. In figure 25 the corresponding shaped input torques and the joint velocities are shown. A resume of the performance is reported in table 12. In this and in the following experiments the computational load was not demanding ($t_s/t_r=0.86$) and could be implemented in real time with the available computing power.

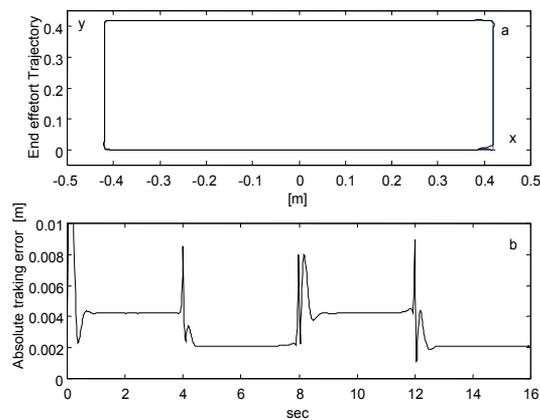


Figure 24. Tracking performance for example 2, experiment 1. Constraints on inputs and angular rates are disabled

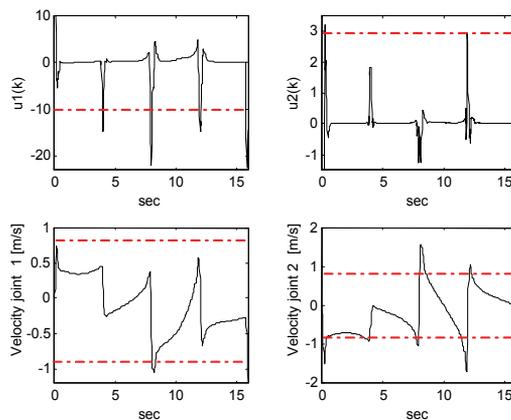


Figure 25. Shaped torques and joints velocities example 2, experiment 1

Case	$ e(k) _{mean}$ (1)	$ e(k) _{max}$ (2)	$ u_1(k) _{max}$ (3)	$ u_2(k) _{max}$ (4)	$ \Delta u_1(k) _{max}$ (5)	$ \Delta u_2(k) _{max}$ (6)	$ \dot{\theta}(k) _{max}$ (7)	$ \dot{\psi}(k) _{max}$ (8)	t_s/t_r (9)	gen/t_s (10)
Exp.1	0.0033	0.0093	22,01	2,93	6	2	1.05	1.76	086	5
Exp.2	0.0056	0.041	10	2	3	0.3	1.383	1.66	0.86	5
Exp.3	0.0087	0.0463	9,84	1.77	3	0.3	0.881	0.900	0.86	5

Table 12. Performance of MPC for $T_c=2T_s$: example 2. Col.1-2 Computing time, Col.3-4 Maximum control effort, Col.5-6 Maximum control increment, Col.7-8 Maximum angular rate, Col.9 Normalized simulation time, Col.10 Generation evaluated in a sampling interval

Experiment 2: In this experiment the constraints (29) and (30) on inputs were activated while the constraint (31) on the velocities remain still inactive ($J_I=0$). Due to torque saturations, an increase in the tracking error is observed particularly in the corners of the trajectory; anyway a good tracking is still achieved. It should be noted that also in this case the constraints on joint velocities are violated. In figure 26a and 26b the trajectory of the tip position and the corresponding absolute tracking error are reported respectively. In figure 27 the shaped input torques and the velocities of joints are shown. A resume of the performance is reported in table 12.

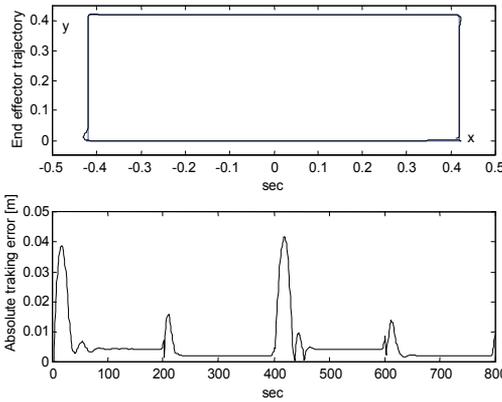


Figure 26. Tracking performance for example 2, experiment 2. Only the constraints on inputs are active

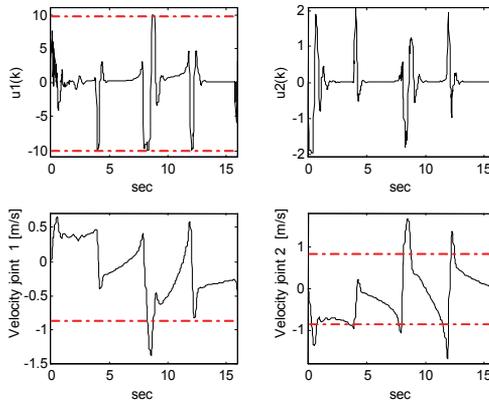


Figure 27. Shaped torques and joints velocities example2, experiment 2

Experiment 3: In the last experiment also the constraints (31) on the velocities have been activated ($J_1 \neq 0$). Due to the presence of these additional constraints, it was not possible to maintain a good tracking in some parts along the trajectory. In figure 28a and 28b the trajectory of the tip position and the corresponding absolute tracking errors are reported. In figure 29 the shaped input torques and the joints velocities are shown. It should be observed that in this case all the constraints were fulfilled; in particular the most decisive constraint was that on the second joint velocity. Indeed, in the periods when this signal is saturated the biggest tracking error was observed. The control torques never reached the saturation. In all the three experiments the trajectory shaper has given an appropriate solution to the constrained robot motion planning problem.

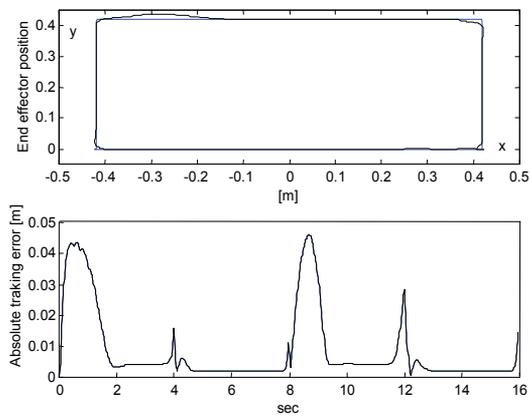


Figure 28. Tracking performance for example 2, experiment 3. Constraints on inputs and angular rates are active

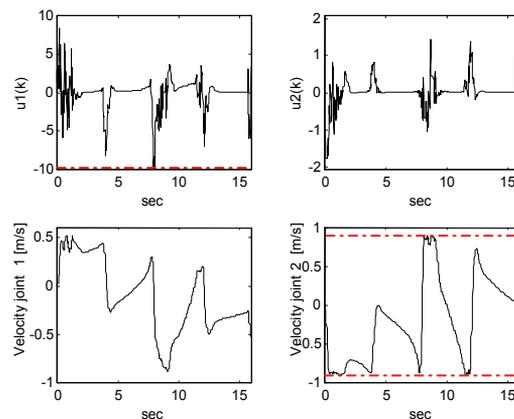


Figure 29. Shaped torques and joints velocities example 2, experiment 3

14. Comparison with conventional optimization methods

The comparison of the Evolutionary optimization for nonlinear MPC with respect to conventional methods was first carried out in (Onnen et Al., 1997); in this study it was showed the superiority of the EA on the branch-and-bound discrete search algorithm. In this study, our intention was to compare the performance provided by the proposed EA with a local gradient-based iterative algorithm. It was implemented a basic gradient steepest descent algorithm and used the standard gradient projection method to fulfill the amplitude and rate constraints for the control commands (Kirk, 1970); the partial derivatives of the index J with respect to the decision variables were evaluated numerically. The performance comparison was performed on the more challenging two-links planar robot system in the case of the experiment 2. Table 13 reports the performance provided by the two methods in terms of the normalized simulation time, the mean absolute tracking error and the number of prediction required by increasing the number of algorithm cycles per sampling interval T_s . The Evolutionary optimization gave remarkably better performance in terms of the mean tracking error. This fact clearly puts into evidence that, in this case, the gradient-based optimization gets trapped in local minima, while the EA provides an effective way to prevent the problem. As for the computational power (N_{sim}/T_s) required by the two methods to converge to a sub optimal solution within 5% of the stationary value, the EA required 160 simulations while the gradient-based 196; the corresponding normalized simulation times were comparable. The possibility of achieving remarkable performance improvement with a comparable computational cost clearly demonstrates that the EA-based predictive shaper offers a valid alternative to iterative local optimization methods especially in the case of multi modal objective functions.

		EA ($N_U=12, N=20$)			GRAD ($N_U=12$)		
$n^\circ\text{-cycles}/T_s$	t_s/t_r	$ e(k) _{mean}$	N_{sim}/T_s	t_s/t_r	$ e(k) _{mean}$	N_{sim}/T_s	
1	0.28	0.0228	20	0.17	0.0311	12	
2	0.45	0.0069	40	0.25	0.0135	24	
4	0.71	0.0057	80	0.40	0.0122	48	
8	1.21	0.0055	160	0.70	0.0115	96	
16	3.10	0.0054	320	1.26	0.0104	192	
32	4.97	0.0054	640	2.39	0.0103	384	

Table 13. MPC performance comparison of EA and gradient-based optimization

15. Conclusions

In this work an online predictive reference shaper has been proposed as a method to improve the tracking performance of robotic systems subjected to constraints on input and state variables. The method allows the choice of an arbitrary multi-objective cost function

that quantifies all the objectives of the desired constrained motion. The global nonlinear optimization is performed in discrete time employing an online specialized Evolutionary Algorithm. The trajectory shaper has been applied in two examples of constrained robotic motion. Both the simulation tests have clearly pointed out the benefits on the performance given by the proposed Predictive Evolutionary Trajectory Shaper; furthermore, the EA algorithm outperformed a conventional iterative gradient-based optimization procedure. It has also been shown that real time implementation can be easily achieved using a not excessively powerful CPU. Finally, it is worth mentioning that the proposed method is sufficiently general and for this reason could be easily extended to other kinds of nonlinear dynamic systems with arbitrary references and constraints.

16. References

- J.E Bobrow, S. Dubowsky, J.S.Gobson, (1985), Time Optimal Control of Robotic Manipulator Along Specified path, *Int. J. Robotics Research*, 4(3), 3-17.
- E.F. Camacho, C. Bordons, (1995) *Model Predictive control in the process industry*, (Berlin: Germany, Springer Verlag.
- W.H. Chen, D.J. Balance, J. O'Reilly, (2000), Model predictive control of nonlinear systems: computational burden and stability, *IEE Proceedings Control Theory and Applications*, 147 (4), , 387 -394.
- O. Dahl, L. Nielsen, (1990) Torque limited path following by online trajectory time scaling, *IEEE Trans. Robotics Automation*, 6(5) 554-561
- B. Foegel, (1994) Applying Evolutionary Programming to Selected Control Problems, *Computers Math. Applic.* 27(11) 89-104
- M. Fischer, O. Nelles, R. Isermann, (1998) Adaptive predictive control of a heat exchanger based on a fuzzy model, *Contr. Eng. Practice* 6(2), , 259-269.
- M.L. Fravolini, A. Ficola, M. La Cava (1999) Improving trajectory tracking by feed forward evolutionary multivariable constrained optimization, *Proc. IEEE/ASME Int. Conf. Advanced Intelligent Mechatronics*, 985-990.
- M.L. Fravolini, A. Ficola, M. La Cava, (1999) Vibration Reduction by Predictive Evolutionary Trajectory Shaping, *IEEE 38th Conference on Decision and Control*, 5289-5291.
- M.L. Fravolini, A. Ficola, M. La Cava (2000) Intermittent Feedback Strategies for the Predictive Control of a Flexible link: Experimental Results, *Proc. 6th IFAC Symposium on Robot Control*, 2000, 391-396.
- C.E. Garcia, D.M. Petit, M. Morari (1989) Model Predictive control: Theory and Practice a survey, *Automatica*, 25(3), 1989, 335-348.
- H. Geniele, R.V. Patel, K. Khorasani (1997), End-point control of a flexible-link manipulator: theory and experiment, *IEEE Trans. Contr. Syst. Technology*, 5(6), 556-570.
- V. Goggos, R.E. King (1996), Evolutionary Predictive Control (EPC), *Computers Chem. Eng.*, 20 suppl. 817-822
- D. Goldberg (1989), *Genetic Algorithms, Search Optimization and Machine Learning*, (Addison Wesley Publishing Company, INC.
- B. Grosman, D.R. Lewin (2002), Automated nonlinear model predictive control using genetic programming, *Computers and Chemical Engineering*, 26, 631-640
- E. Gyurkvcis (1998), Receding horizon control via bolza-type optimization, *Sys. & Contr. Letters*, 35, 195-200.
- D.E. Kirk (1970), *Optimal Control Theory*, Prentice-Hall.

- K. Kumagai, D. Kohli, R. Perez, (1996) Near minim time feedback controller for manipulators using online time scaling of trajectories, *ASME J. Dynamic Sys. Meas. Contr.*, 118, 300-308
- W.K. Lennon, K.M. Passino, (1999) Intelligent Control for brake systems, *IEEE Trans. Contr. Sys. Tech.* 7(2), 188-202.
- W.S. Levine, (1996) *The control handbook*, CRC Press.
- D.C. Liaw, J.T. Huang, (1998) Contact Friction Compensation for Robots Using Genetic Learning, *J. of Intelligent and Robotic Systems*, 23(2/4), 331-349.
- D.A. Linkens, H.O. Nyongesa, (1995) Genetic Algorithms for Fuzzy Control part 2: Online system development and application, *IEE Proc. Control Theory Appl.*, 142(3), 177 - 185.
- R. Luus, (1990) Optimal control by dynamic programming using a systematic reduction of the grid size, *Int. J. Control*, 51, 995-1013.
- D.Q. Maine, H. Michalaska, (1993) Robust receding horizon control of nonlinear systems, *IEEE Trans. Automatic Control*, 38, 1623-1633.
- M. Martinez, J. Senent, X. Blasco, (1998) Generalized predictive control using genetic algorithms, *Eng. Application of Artificial Intelligence*, 11(3), 1998, 355-367.
- R.K Mutha, W.R Cluett, A. Penlidis (1997), Nonlinear model-based predictive control of non affine systems, *Automatica*, 33(5), 907-913.
- C. Onnen , R. Babuska, U. Kaymak, J.M. Sousa, H.B. Verbruggen, R. Iserman, (1997), Genetic algorithms for optimization in predictive control, *Contr. Eng. Practice*, 5(10), 1997, 1363-1372.
- L.L. Porter, K.M. Passino (1995), Genetic adaptive observers, *Eng. Application. of Artificial Intelligence*, 8(3), 261-269.
- L.L. Porter, K.M. Passino, (1998) Genetic Adaptive and Supervisory control, *Int. J. Intelligent Control Sys*, 2(1), 1-41.
- A.S. Rana, A.M.S. Zalzal, (1996) Near time optimal collision used free motion planning of robotic manipulators using evolutionary Algorithm, *Robotica*, 14 621-632
- E. Ronco, T. Arsan, P.J. Gawthrop, D.J. Hill, (1999) A globally valid continuous-time GPC through successive system linearisation, *Proc. 38th IEEE Conf. Decision and Control*, 726-731.
- E. Ronco, T. Arsan, P.J. Gawthrop, (1999) Open loop intermittent feedback control: practical Continuous-time GPC, *IEE Proceedings* 146(5), 426-434.
- J.A. Roubos, S. Mollow, R. Babuska, H.B. Verbruggen (1999), Fuzzy model based predictive control using Takagi-Sugeno models, *Int. J. Approximate Reasoning* , 22(1-2), 1-30.
- Z. Shiller, W.S. Chang (1995), Trajectory preshaping for high speed articulated systems, *J. of Dynamic Sys. Meas. and Control*, 117, 304-310.
- P.O.M. Scokaert, D.Q. Maine, J.B. Rawings (1999), Suboptimal model predictive control (feasibility implies stability), *IEEE Trans. Automatic Control*, 44(3), 648-658.
- Z. Shin, N.D. Mc Kay, (1985) Minimum time control of robotic manipulators with geometric path constraints, *IEEE Trans. Robot. Automation*, 30 531-541
- S.C. Shin, S.B. Park, (1998), GA based predictive control for nonlinear processes, *Electronic Letters*, 34(20), , 1980-1981.
- B.J Song, A.J. Koivo, (1999), Nonlinear predictive control with application to manipulator with flexible forearm, *IEEE Trans. Ind. Electr.*, 46(5), 923 -932.

- J.M. Sousa, M.S. Setnes, (1999), Fuzzy predictive filters in model predictive control, *IEEE Trans. Ind. Electr.* 46(6), 1225 -1232.
- S. Sun, A.S. Morris, A.M.S. Zalzala, (1996) Trajectory planning of multiple coordinating robots using genetic algorithms, *Robotica*, 14 227-234
- L. Yao, W. Sethares, (1994) Nonlinear parameter estimation via the genetic algorithm, *IEEE Trans. on Sign. Proc.*, 42(4), 927 -935.

Applying Real-Time Survivability Considerations in Evolutionary Behavior Learning by a Mobile Robot

Wolfgang Freund, Tomás Arredondo V. and César Muñoz
*Universidad Técnica Federico Santa María, Departamento de Electrónica
Chile*

1. Introduction

In this chapter we investigate real-time extensions for evolutionary mobile robot learning. The learning performance is measured in navigation experiments of complex environments as performed in a Kephra mobile robot simulator (YAKS). All these experiments are done in the context of our recently introduced motivation based interface that provides an intuitive human-robot communications mechanism (Arredondo et al., 2006). This motivation interface has been used in a variety of behavior based navigation and environment recognition tasks (Freund et al., 2006).

Our first heuristic introduces active battery level sensors and recharge zones, which are used as soft deadlines to improve robot behavior for reaching survivability in environment exploration. Based on our previously defined model, we also propose a hard deadline based hybrid controller for a mobile robot, combining behavior-based and mission-oriented control mechanisms.

These methods are implemented and applied in action sequence based environment exploration tasks in a YAKS mobile robot simulator. We validate our techniques with several sets of configuration parameters on different scenarios. We consider soft-deadlines as a dangerous but not critical battery charge level which affect a robot's fitness. Hard-deadlines are considered as a possible (because of partial knowledge) point where, if the robot does not recharge his battery, an unrecoverable final freezing state is possible. Our tests include action sequence based environment exploration tasks. These experiments show a significant improvement in robot responsiveness regarding survivability and environment exploration when using these real-time based methods.

The rest of the chapter is organized as follows. In Section 2 a description of our soft-computing based navigation model is given. In Section 3 real-time extensions of our model are presented. In Section 4 we show the experimental setup and test results. Finally, in Section 5 some conclusions and future work are drawn.

2. Soft-computing Based Robotic Navigation

Navigation and environment recognition is something that is taken for granted by most of us but as we see when observing an infant this is a difficult task fraught with peril for the

inexperienced navigator. Thanks to its tolerance of imprecision and incomplete data, soft-computing has had much recent success in robotic navigation where other more structured or formal methods have not fared so well (Arkin, 1998; Jang et al., 1997; Konolige et al. 1992; Goodridge, 1997 ; Kem & Woo, 2005). Fuzzy logic has been a mainstay of several such efforts: applying fuzzy rule based method for a hexapod obstacle avoidance (Kem & Woo, 2005), using independent distributed fuzzy agents and weighted vector summation via fuzzy multiplexers for producing drive and steer commands (Goodridge, 1997), neuro-fuzzy controllers for behavior design (Hoffmann, 2000), fuzzy modular motion planning (Al-Khatib & Saade, 2003), fuzzy integration of groups of behaviors (Izumi & Watanabe, 2000), multiple fuzzy agents for behavior fusion (Barberá & Skarmeta, 2002), GA based neuro fuzzy reinforcement learning agents (Zhou, 2002), and fuzzy logic integration for robotic navigation in challenging terrain (Seraji & Howard, 2002).

Using fuzzy logic as a motivation toward a variety of useful behaviors is something that has not seen wide usage in robotics previously. Toward this goal, we have implemented such motivations (e.g. a need, desire or want) as fuzzy fitness functions that serve to influence the intensity and direction of robotic behaviors (Arredondo et al., 2006). In general, motivations are accepted as involved in the performance of learned behaviors given that a learned behavior may not occur unless it's driven by a motivation (Huitt, 2001). Having a variety of motivations helps produce diverse behaviors some of which have a high degree of benefit (or fitness) for the organism.

In our experiments, we have used fuzzy membership functions (Fig. 1) toward implementing a motivation based interface for determining robotic fitness. There are five triangular functions used for each of the four motivations in our experiment (Very Low, Low, Medium, High, Very High).

The motivation set (M) considered in this study includes: curiosity (m_1), homing (m_2), and energy (m_3). These motivations are used as input settings (between 0 and 1) prior to running each experiment.

During training, a run environment (room) is selected and the GA initial robot population is randomly generated. After this, each robot in the population performs its navigation task and a set of fitness values corresponding to the performed task are obtained (f_1 through f_3). Finally, robotic fitness is calculated using the fitness values information provided by the simulator and the different motivations at the time of exploration (Fig. 2).

Takagi-Sugeno-Kang (TSK) fuzzy logic model is used, TSK fuzzy logic does not require defuzzification as each rule has a crisp output that is aggregated as a weighted average (Jang et al., 1997).

The membership functions used are given in Fig. 1. Sample fuzzy rules (numbers 9 and 10) are given as follows:

if ($f_1 == M$) and ($f_2 == V.L.$) and ($f_3 == V.L.$) then

$$f[9] = m_1f_1C[3]+m_2f_2C[1]+m_3f_3C[1]$$

if ($f_1 == M$) and ($f_2 == V.L.$) and ($f_3 == L$) then

$$f[10] = m_1f_1C[3]+m_2f_2C[1]+m_3f_3C[1]$$

The values for these fitness criteria are normalized (range from 0 to 1). The criteria and the variables that correspond to them are: amount of area explored (f_1), proper action termination and escape from original neighborhood area (f_2), and percent of battery usage (f_3). These fitness values are calculated after the robot completes each run. The f_1 value is

determined by considering the percentage area explored relative to the optimum, f_2 is determined by $f_2 = 1 - l/L$, where l is the final distance to robot's home and L the theoretical maximum value. Finally f_3 is the estimated total energy consumption of the robot considering each step.

The final fuzzy motivation fitness value (F) is calculated using TSK based fuzzy logic (three fuzzy variables with five membership functions each: $3^5 = 243$ different fuzzy rules) as shown in Fig. 3 and using the membership functions to compute μ , values. For the coefficient array C we used a linear function.

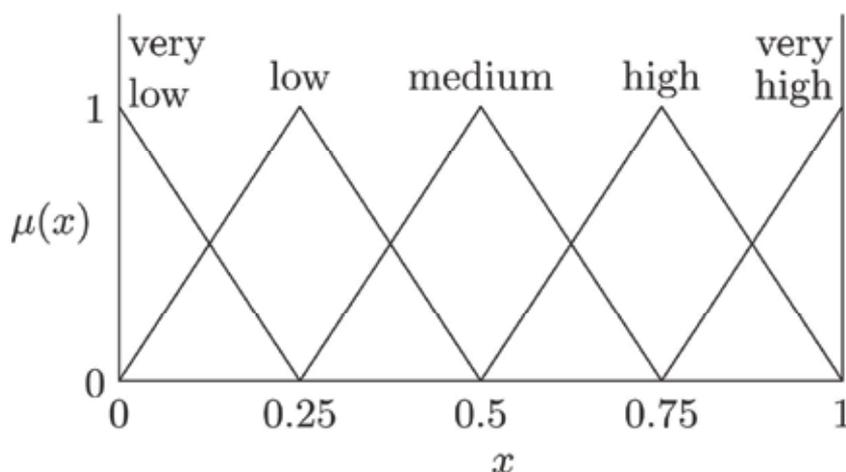


Figure 1. Fuzzy membership functions

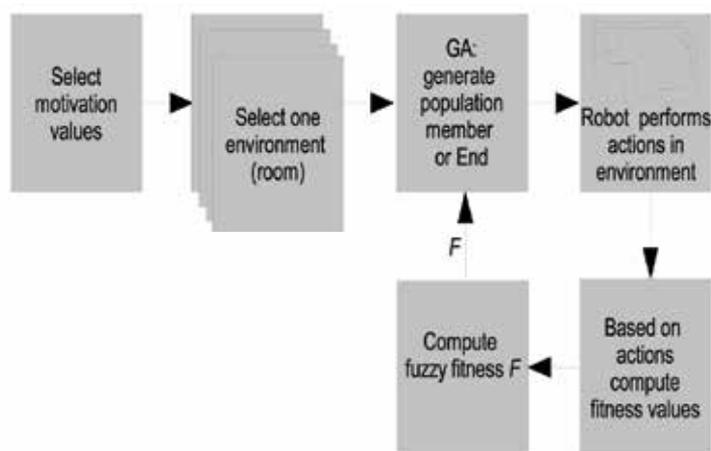


Figure 2. System overview

Algorithm FuzzyFitness

Input:

N : number of fuzzy motivations;
 M : number of membership functions per motivation;
 $X[N]$: array of motivation values preset;
 $Y[N]$: array of fitness values;
 $C[N]$: array of coefficients;
 $\mu[N][M]$: matrix of membership values for each motivation;

Variables:

$w[n]$: the weight for each fuzzy rule being evaluated;
 $f[n]$: the estimated fitness;
 n, x_0, x_1, \dots, x_N : integers;

Output:

F : the fuzzy fitness value calculated;

begin

$n := 1$;

for each $x_1, x_2, \dots, x_N := 1$ **step 1 until** M **do**

begin

$w[n] := \min\{\mu[1][x_1], \mu[2][x_2], \dots, \mu[N][x_N]\}$;

$f[n] := \sum_{i=1}^N X[i]Y[i]C[x_i]$;

$n := n + 1$;

end;

$F := (\sum_{i=1}^{NM} w[i]f[i]) / (\sum_{i=1}^{NM} w[i])$;

end;

Figure 3. Fuzzy fitness algorithm

2.1 Implementation

The YAKS (Yet Another Khepera Simulator) simulator is the base for our implementation. YAKS is a simple open source behavior-based simulator (YAKS) that uses neural networks and genetic algorithms to provide a navigation environment for a Khepera robot as seen in Fig. 5. Sensors are directly provided into a multilayer neural network in order to drive left and right wheel motors (Fig. 4). A simple genetic algorithm is used with 200 members, 100 generations, mutation of 1%, and elite reproduction. Random noise (5%) is injected into sensors to improve realism. The GA provides with a mechanism for updating neural network weights used by each robot in the population that is being optimized. An overview of our fuzzy fitness implementation is shown in Fig. 6.

Outputs of the Neural Network are real valued motor commands (Motor_L and Motor_R) between 0 and 1 which are discretized into one of four actions (left 30°, right 30°, turn 180°, go straight). This follows the Action-based environmental modeling (AEM) search space reduction paradigm (Yamada, 2005).

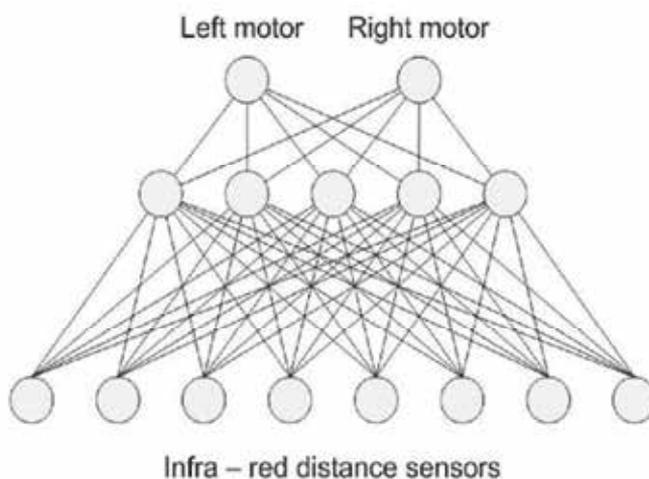


Figure 4. Robotic neural network

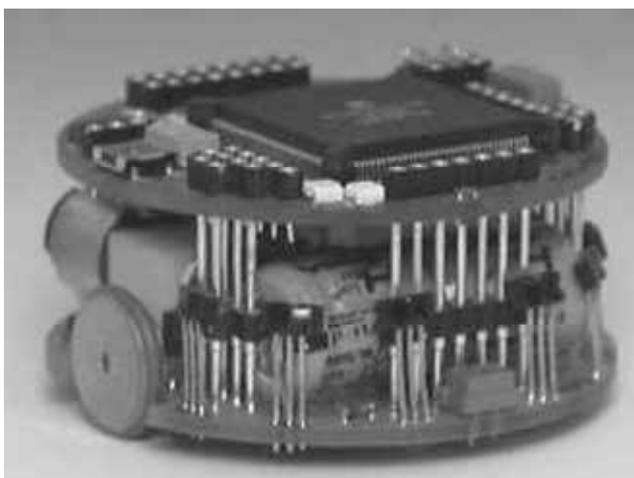


Figure 5. Kephra robot

3. Real-Time Extensions

Real-time systems are concerned with real-world applications, where temporal constraints are part of system specification imposed by the environment, i.e. firm-deadlines in QOS environments, soft-deadlines in non-critical control applications and hard deadlines in safety-critical systems. In the last years more research effort have been made applying soft-computing techniques to real-time control problems (Wang & Lee, 2003; Jha et al., 2005; Seraji & Howard, 2002; Maione & Naso, 2003; Ziemke et al., 2005). The main advantage over traditional control mechanisms is in the additional robustness regarding lack or poor environmental information (if not the problem definition itself) which concerns almost all real-time control applications (Sick et al., 1998).

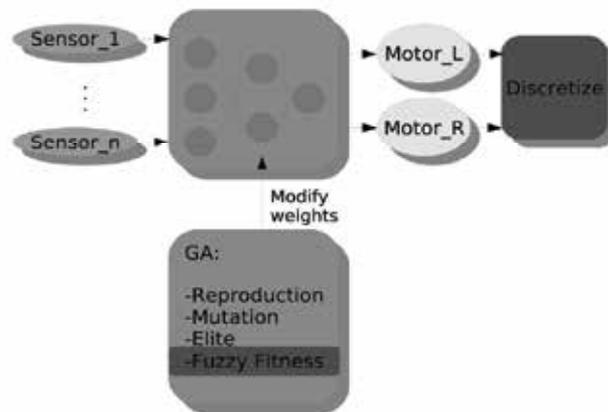


Figure 6. Robotic system implementation

On the other hand, soft-computing based methods are more intuitive than strict formal models, soft-computing (e.g. fuzzy logic) aims to gain from operator perceptions and through iterative improvements in the associated rule set tries to obtain the capabilities of the real expert. However, not much attention has been given to real-time considerations, regarding soft or hard deadlines. Some important aspects of real-time must be taken into account: how could soft-computing techniques, such as fuzzy logic, neural networks or genetic algorithms affect systems responsiveness and survivability?

In order to introduce our behavior based mobile robot methodology in a real-world application, we introduce an active battery sensor to allow for the detection of low battery conditions and we also provide various number of recharge zones within different room configurations. These real-time extensions must be capable of supporting different sets of motivations while also improving survivability and exploration performance.

Our primary goal consists of full environment exploration considering energy consumption and recharge zones. To reach this objective, the robot's behavior must be influenced through periodic energy evaluation for recharging the battery before it is too late. In this approach we consider soft-deadlines as a dangerous but not critical battery charge level which affect robot's fitness. Hard-deadlines are considered as a possible (because of partial knowledge) point where, if the robot does not recharge his battery, an unrecoverable final freezing state is possible. Soft vs hard-deadlines force a change in the robot's operation from behavior-based to mission-oriented (hybrid), which guides the robot using the shortest known path to a nearest previously found charging zone.

During environment exploration, autonomous or semi-autonomous mobile robots are confronted with events which could be predictable such as walls and static objects, or unpredictable such as moving objects or environmental changes. Some of these events must be attended in real-time (responsiveness) to guarantee the robot's integrity (survivability) (Kopetz, 1997).

Traditional control mechanisms are based on reliable real-time systems, i.e. time constraints over executions and predictability (Gheith & Schwan, 1993), also known as dependable systems (Motet & Geffroy, 2003), e.g. the mars pathfinder or DUSAUV, a semi-autonomous underwater vehicle presented in (Li et al., 2005). On the other hand, soft-computing based methods have not been widely used in this arena due to their inherent uncertainty.

In order to introduce real-time considerations into our behavior-based mobile robot for a real-world application, we extend our model by using temporal constraints during the navigation test-phase. The constraints considered include energy consumption and finite battery charge capacity.

In our approach soft-deadlines dynamically affects the robots navigation. This could influence behaviors to avoid highly energy consuming actions and could guide the robot's movement to a recharging zone as necessary.

When a critical battery level is reached, the previously defined method is no longer useful. A responsive real-time method is needed to, if possible, guarantee survivability (Kopetz, 1997). Strictly speaking, we can't guarantee survivability because of the robots partial knowledge of the world map which, initially, has no recharge zones mapped (we do not consider the starting point as a recharging zone). Nevertheless, because of the off-line robot training-phase, we expect that the trained robot (e.g. NN) will be capable of finding charging zones. Using the charge zone information obtained on-line, the robot applies real-time based navigation. We establish a hard-deadline as the point of the robot's unrecoverable final freezing state. Before reaching this deadline (with a 10% safety margin) the robot's operation mode changes from behavior-based to mission oriented, following the shortest path to the nearest previously found charging zone (Tompkins et al., 2006).

4. Experimental Evaluation

The major purpose of the experiments reported in this section is to study the influence of our real-time extensions over the robot's behavior, considering survivability and exploration capability.

We have designed four different rooms (environments) for the robot to navigate in. We denote these rooms as: S-ROOM (the simplest), L-ROOM, T-ROOM and H-ROOM (most complex). Walls are represented by lines and we designate up to three charging zones (see circles in Fig. 7). The starting zones for each room will be the lower left corner (quarter circles in Fig. 7).

We will denote by NRT the behavior-based algorithm which operates the robot without any real-time considerations, i.e. the battery level has no influence over robot's behavior but, if it comes near to a charging zone the battery level is updated to his maximum capacity. The main characteristics of NRT are:

- the battery level has no influence on the robot during training phase and,
- there is no input neuron connected to the battery sensor.

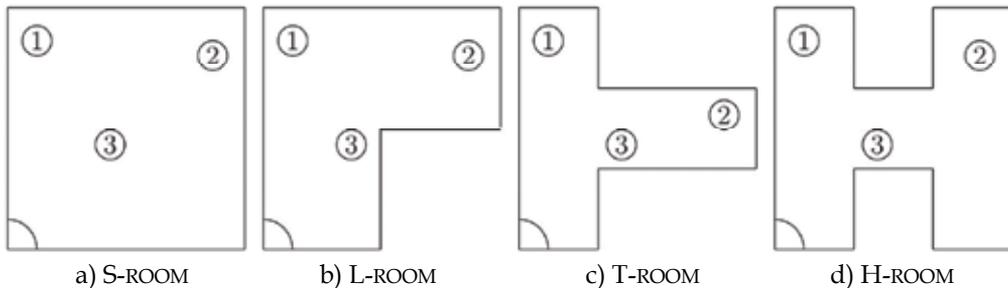


Figure 7. Experiment rooms layout with starting and recharging zones

We denote by SRT the algorithm which operates the robot with soft-real time considerations, influencing his behavior to avoid a dangerous battery level. This algorithm differs from NRT mainly by

- battery level influences robot's fitness evaluation used by the GA and,
- a new input neuron is connected to a battery level sensor.

Finally, we denote by HRT the hybrid algorithm which operates the robot with hard-real time considerations, i.e., the same as SRT incorporating critical battery level sensing, and also having the capacity to change the robot's normal operation to mission oriented, guaranteeing his survivability (if at least one charging zone was previously found).

4.1 Experimental Setup

As mentioned before, the experiments are performed using a modified version of YAKS. This simulation system has several different elements including: the robot simulator, neural networks, GA, and fuzzy logic based fitness.

Khepera Robot For these simulations, a Khepera robot was chosen. The robot configuration has two DC motors and eight (six front and two back) infrared proximity sensors used to detect nearby obstacles. These sensors provide 10 bit output values (with 5% random noise), which allow the robot to know in approximate form the distance to local obstacles. The YAKS simulator provides the readings for the robot sensors according to the robot position and the map (room) it is in. The simulator also has information for the different areas that the robot visits and the various obstacles (walls) or zones (home, charging zones) detected in the room. In order to navigate, the robot executes up to 1000 steps in each simulation, but not every step produces forward motion as some only rotate the robot. If the robot has no more energy, it freezes and the simulation stops.

Artificial Neural Network The original neural network (NN) used has eight input neurons connected to the infrared sensors, five neurons in the hidden layer and two output neurons directly connected to the motors that produce the robot movement. Additionally, in our real-time extensions we introduce another input neuron connected to the battery sensor (activated by SRT and HRT).

Genetic Algorithm A GA is used to find an optimal configuration of weights for the neural network. Each individual in the GA represents a NN which is evolving with the passing of different generations. The GA uses the following parameters:

- Population size: 200
- Crossover operator: random crossover
- Selection method: elite strategy selection
- Mutation rate: 1%
- Generations: 100

For each room (see Fig. 7) we trained a robot up to 400 steps, considering only configurations with 2 or 3 charging zones, i.e. shutting down zone 3 for 2-zones simulations. Startup battery level allows the robot to finish this training phase without recharging requirements.

Finally, we tested our algorithms in each room up to 1000 steps, using the previously trained NN for each respective room. The startup battery level was set to 80 (less than 50% of it's capacity), which was insufficient to realize the whole test without recharging.

4.2 Experimental Results

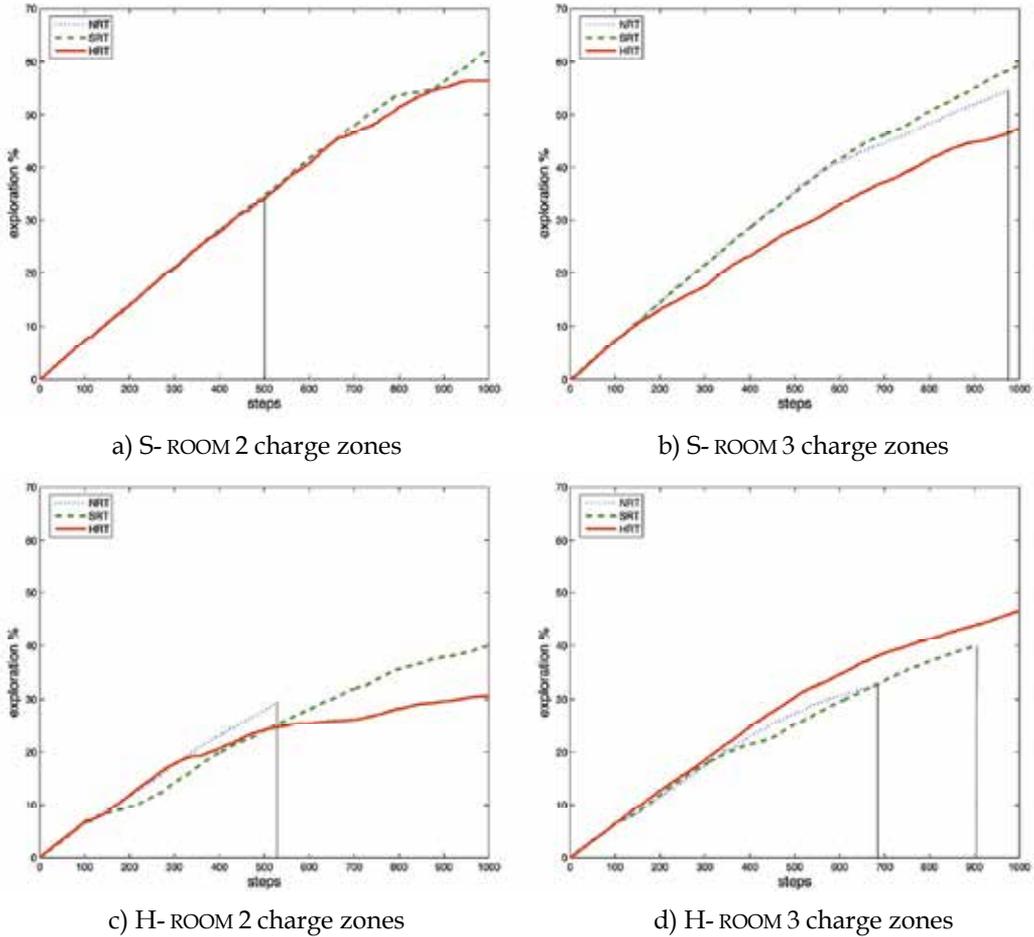


Figure 8. Exploration behaviour

We chose the S-ROOM and H-ROOM to show results for a simple and complex room respectively, which are representative behaviors of our approach.

In Fig. 8 we show the robot's exploration behavior for selected rooms. Each curve in the graph shows the average value of 10 executions of the same experiment (deviation between experiment iterations was very small, justifying only 10 executions). Let $surv(a)_i$ the survivability of the experiment instance i of algorithm a , we define $surv(a)$ as the survivability of an experiment applying algorithm a as the worst case survivability instance of an experiment, i.e.

$$surv(a) = \min_{i=1, \dots, 10} [surv(a)_i] \quad (1)$$

Please note that the end of each curve in Fig. 8 denotes the survivability of the respective algorithm (for better readability, we mark NRT survivability with a vertical line). Reaching step 1000 (the maximum duration of our experiments) means that the robot using the

algorithm survives the navigation experiment. Finally, in Fig. 9 we show a representative robot's battery level. Monitoring was made during test phase in a H-ROOM with 3 charging zones.

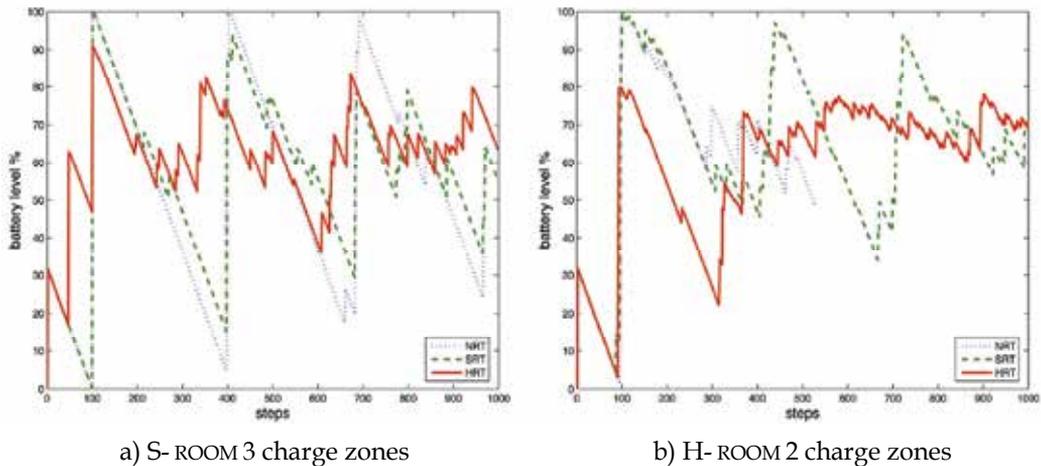


Figure 9. Battery Behaviour

4.3 Discussion

The results of our experiments are summarized below:

Survivability: As shown in Fig. 8, SRT and HRT algorithms give better reliability of completing missions than the NRT method, independently of the rooms (environments) we use for testing (see Fig. 7). As expected, if fewer charging zones are provided, NRT has a less reliable conduct. Please note that as shown in Fig. 9, NRT is also prone to battery depletion risk and does not survive in any case.

When varying the room complexity, i.e. 8(b) and 8(d), real-time considerations have significant impact. Using SRT, a purely behavior-based driven robot (with the additional neuron and motivation), improves its performance. The SRT method does not guarantee Survivability since without changing the robot operation from behavior based to mission oriented the robot is prone to dying even with a greater number of recharge zones (as seen in 8(d)). Finally, we conclude that despite the uncertainty introduced by soft-computing methods, HRT (e.g. the hybrid algorithm), in general is the best and safest robot control method from a real-time point of view.

Exploration Environment: As can be seen in Fig. 8 safer behaviors means slower exploration rates (more conservative), up to 12% slower in our experiments. When comparing NRT with SRT, the exploration rates are almost equal in simple environments. In more complex rooms, SRT exploration is slower than NRT (due to battery observance). However, because of SRT having better survivability on the whole its performance wins over NRT. If we compare NRT with HRT, exploration performance also favors NRT, which could be explained given HRT conservative battery management (see Fig. 9).

Given 2 charge zones, HRT behaves differently in environments of varying complexity (up to 25%) which could be attributed to the complexity of the returning path to the nearest charging zone and losing steps in further exploration. This phenomena becomes less notorious when increasing the number of charging zones (more options for recharge).

5. Conclusions and Future Work

In this work we investigate real-time adaptive extensions of our fuzzy logic based approach for providing biologically based motivations to be used in evolutionary mobile robot learning. We introduce active battery level sensors and recharge zones to improve robot's Survivability in environment exploration. In order to achieve this goal, we propose an improvement of our previously defined model (e.g. SRT), as well as a hybrid controller for a mobile robot (e.g. HRT), combining behavior-based and mission-oriented control mechanisms.

These methods are implemented and tested in action sequence based environment exploration tasks in a Khepera mobile robot simulator. Experimental results shows that the hybrid method is in general, the best/safest robot control method from a real-time point of view. Also, our preliminary results shows a significant improvement on robot's survivability by having minor changes in the robot's motivations and NN.

Currently we are implementing a real robot for environment exploration to validate our model moving from simulation to experimentation. We are also introducing dynamic motivations schedules toward robotic behavior enhancement. Improving the dependability of HRT, we want to extend this control algorithm to safety-critical domains.

6. References

- T. Arredondo, W. Freund, C. Muñoz, N. Navarro, & F. Quirós. (2006). Fuzzy motivations for evolutionary behavior learning by a mobile robot. *Lecture Notes in Artificial Intelligence*, 4031:462-471.
- Mohannad Al-Khatib & Jean J. Saade. (2003). An efficient data-driven fuzzy approach to the motion planning problem of a mobile robot. *Fuzzy Sets Syst.*, 134(1):65-82.
- Ronald C. Arkin. (1998). Behavior-Based Robotics. MIT Press.
- Humberto Martínez Barberá & Antonio Gómez Skarmeta. (2002). A framework for defining and learning fuzzy behaviors for autonomous mobile robots. *International Journal of Intelligent Systems*, 17(1):1-20.
- W. Freund, T. Arredondo, C. Muñoz, N. Navarro, & F. Quirós. (2006). Realtime adaptive fuzzy motivations for evolutionary behavior learning by a mobile robot. *Lecture Notes in Artificial Intelligence*, 4293:101-111.
- S. Goodrige, M. Kay, & R. Luo. (1997). Multi-layered fuzzy behavior fusion for reactive control of an autonomous mobile robot. In *Proceedings of the Sixth IEEE International Conference on Fuzzy System*, pages 573-578, July 1997.
- Ahmed Gheith & Karsten Schwan. (1993). Chaosarc: kernel support for multiweight objects, invocations, and atomicity in real-time multiprocessor applications. *ACM Transactions on Computer Systems*, 11(1):33-72, February 1993.
- Frank Hoffmann. (2000). Soft computing techniques for the design of mobile robot behaviors. *Inf. Sci.*, 122(2-4):241-258.
- W. Huitt. (2001) Motivation to learn: An overview. Technical report, Educational Psychology Interactive, Valdosta State University.
- Kiyotaka Izumi & Keigo Watanabe. (2000). Fuzzy behavior-based control trained by module learning to acquire the adaptive behaviors of mobile robots. *Math. Comput. Simul.*, 51(3-4):233-243.

- J.-S. R. Jang, C.-T. Sun, & E. Mizutani. (1997). *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*. NJ: Prentice-Hall. ISBN: 0-13-261066-3.
- Rahul Kumar Jha, Balvinder Singh, & Dilip Kumar Pratihari. (2005). On-line stable gait generation of a two-legged robot using a genetic-fuzzy system. *Robotics and Autonomous Systems*, 53(1):15–35, October 2005.
- Kurt Konolige, Karen Meyers, and Alessandro Saffiotti. (1992). Flakey, an autonomous mobile robot. Technical report, Stanford Research Institute International, July 1992.
- Hermann Kopetz. (1997). *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Norwell, MA, USA.
- Makoto Kern & Peng-Yung Woo. (2005). Implementation of a hexapod mobile robot with a fuzzy controller. *Robotica*, 23(6):681–688.
- Ji-Hong Li, Bong-Huan Jun, Pan-Mook Lee, & Seok-Won Hong. (2005). A hierarchical real-time control architecture for a semi-autonomous underwater vehicle. *Ocean Engineering*, 32(13):1631–1641, September 2005.
- G. Motet & J.-C. Geffroy. (2003). Dependable computing: an overview. *Theor. Comput. Sci.*, 290(2):1115–1126, 2003.
- Guido Maione & David Naso. (2003). A soft computing approach for task contracting in multi-agent manufacturing control. *Comput. Ind.*, 52(3):199–219.
- H. Seraji & A. Howard. (2002). Behavior-based robot navigation on challenging terrain: A fuzzylogic approach. *IEEE Transactions on Robotics and Automation*, 18(3):308–321.
- Homayoun Seraji & Ayanna Howard. (2002). Behavior-based robot navigation on challenging terrain: A fuzzy logic approach. *IEEE Transactions on Robotics and Automation*, 18(3):308–321, June 2002.
- Bernhard Sick, Markus Keidl, Markus Ramsauer, and Stefan Seltzsam. (1998). A Comparison of Traditional and Soft-Computing Methods in a Real-Time Control Application. In *ICANN 98, Proc. of the 8th Int. Conference on Artificial Neural Networks*, pages 725–730, Sweden, September 1998.
- Paul Tompkins, Anthony Stentz, & David Wettergreen. (2006). Mission-level path planning and re-planning for rover exploration. *Robotics and Autonomous Systems*, 54(2):174–183, February 2006.
- Jeen-Shing Wang & C.S. George Lee. (2003). Self-adaptive recurrent neurofuzzy control of an autonomous underwater vehicle. *IEEE Transactions on Robotics and Automation*, 19(2):283–295, April 2003.
- YAKS simulator website: <http://r2d2.ida.his.se/>.
- S. Yamada. (2005). Evolutionary behavior learning for action-based environment modeling by a mobile robot. *Applied Soft Computing*, 5(2):245–257, January 2005.
- C. Zhou. (2002). Robot learning with ga-based fuzzy reinforcement learning agents. *Information Sciences*, 145(1), August 2002.
- Tom Ziemke, Dan-Anders Jirnhed, & Germund Hesslow. (2005). Internal simulation of perception: a minimal neuro-robotic model. *Neurocomputing*, 68:85–104.

An Evolutionary MAP Filter for Mobile Robot Global Localization

L. Moreno¹, S. Garrido¹, M. L. Muñoz², and D. Blanco¹

¹ *Robotic's Laboratory, Universidad Carlos III, Madrid*

² *Facultad de Informática, Universidad Politécnica, Madrid
Spain*

1. Introduction

This chapter presents a new evolutionary algorithm for Mobile Robot Global Localization. The Evolutive Localization Filter (ELF) presented in this paper is a non linear filter algorithm which is able to solve the global localization problem in a robust and efficient way. The proposed algorithm searches along the configurations space for the best robot pose estimate. The elements of each generation are the set of pose solutions and represent the areas with more probability according to the perception and motion information up to date. The population evolves according to the observation and the motion error derived from the comparison between observed and predicted data obtained from the probabilistic perception and motion model. The algorithm has been tested using a mobile robot with a laser range finder to demonstrate the effectiveness, robustness and computational efficiency of the proposed approach.

Mobile robot localization finds out a robot's coordinates relatives to its environment, assuming that one is provided with a map of the environment. Localization is a key component in navigation and required to execute successfully a trajectory. We can distinguish two different cases: the re-localization case and the global localization case. Re-localization or tracking problem tries to keep track of mobile robot's pose, where the robot knows its initial position (at least approximately) and therefore has to maintain localized the robot along the given mission. The global localization problem does not assume any knowledge about the robot's initial position and therefore has to globally localize itself.

The two most important aspects that have to be dealt with when designing a localization system is how to represent uncertain information of the environment and the robot's pose. Among the many ways to represent the knowledge about an environment, this article deals with geometrical localization methods and assumes that the environment is modelled geometrically as an occupancy grid map.

In the robot's pose uncertainty representation and estimation techniques, the vast majority of existing algorithms address only the position tracking problem. In this case the small incremental errors produced along the robot motion and the initial knowledge of the robot's pose makes classical approaches such as Kalman filters or Scan Matching techniques applicable. If we consider the robot's pose estimation as a Bayesian recursive problem, Kalman filters estimate posterior distribution of robot's poses conditioned on sensor data.

Based on the Gaussian noise assumption and the Gaussian-distributed initial uncertainty this method represents posteriors distributions by Gaussians. Kalman filter constitutes an efficient solution for re-localization problems. However, the assumption's nature of the uncertainty representation makes Kalman filters not robust in global localization problems. Scan Matching techniques are also an iterative local minimization technique and can not be used for global localization.

Different families of algorithms can solve the global localization problem, some frequently used are: multi-hypothesis Kalman filters, grid-based probabilistic filters and Monte Carlo localization methods. Those methods can be included in a wider scope group of Bayesian estimation methods. Multi-hypothesis Kalman filters (Arras et al., 2002, Austin & Jensfelt, 2000, Jensfelt & Kristensen, 1999, Cox & Leonard, 1994, Roumeliotis et. al, 2000) represent distributions using mixtures of Gaussians, enabling them to keep track of multiple hypothesis, each of which is represented by a separate Gaussian. This solution presents some initialization problems: one of them is the determination of the initial hypotheses (the number can be very high and it is not bounded), this leads the algorithm to a high computational cost at initial stages. Besides, the Kalman filter is essentially a gradient based method and consequently poorly robust if the initial hypothesis is bad or noise assumptions fails. Grid-based localization algorithms (Fox et al., 1999, Burgard et al., 1996, Reuter, 2000) represent distributions by a discrete set of point probabilities distributed over the space of all possible poses. This group of algorithms are capable of representing multi-modal probabilities distributions. A third group is the Monte Carlo localization algorithms (Jensfelt et al., 2000, Thrun et al., 2001, Dellaert et al., 1999). These algorithms represent the probability distribution by means of a set of samples drawn according to the posterior distribution over robot's poses. These algorithms can manage arbitrary noise distributions and non-linearities in the system and observation models. These methods present a high computational cost due to its probabilistic nature requires a high number of samples to draw properly the posterior probability density function. The main advantage is its statistical robustness.

This article presents a localization algorithm based on a non-linear filter called Evolutionary Localization Filter (ELF). ELF solves the global localization robot problem in a robust and efficient way. The algorithm can deal with arbitrary noise distributions and non-linear space state systems. The key idea of ELF is to represent the uncertainty about the robot's pose by a set of possible pose estimates weighted by a fitness function. The state is recursively estimated using set of solutions selected according on the weight associated to each possible solution included in the set. The set of solutions evolve in time to integrate the sensor information and the robot motion information. The adaptation engine of the ELF method is based on an evolutive adaptation mechanism which combines a stochastic gradient search together with probabilistic search to find the most promising pose's candidates.

2. Differential evolutionary filter

The evolutive optimization techniques constitute a series of probabilistic search methods that avoid derivatives or probability density estimations to estimate the best solution to a localization problem. In the method proposed here each individual in the evolutive algorithm will represent a possible solution to the localization problem and the value of the loss function will represent the error to explain the perceptual and motion data. The search of this solution is done stochastically employing an evolutive search technique based on the

differential evolution method proposed by Storn and Price (Storn & Price, 2001) for global optimization problems over continuous spaces. The Evolutive Filter uses a parallel direct search method which utilizes n dimensional parameter vectors $x_i^k = (x_{i,1}^k, \dots, x_{i,n}^k)^T$ to point each candidate solution i to the optimization problem at iteration step k . This method utilizes N parameter vectors $\{x_i^k; i = 0, 1, \dots, N\}$ as a population for each generation t of the optimization process. Each element of the population set represents a possible solution, but it hasn't associated a probability value to each one (in the particle filter case, each element of the particle set has associated a probability value).

The initial population is chosen randomly to cover the entire parameter space uniformly. In absence of a priori information the entire parameter space has the same probability of containing the optimum parameter vector, and a uniform probability distribution is assumed. The differential evolution filter generates new parameter vectors by adding the weighted difference vector between two population members to a third member. If the resulting vector yields a lower objective function value than a predetermined population member, the newly generated vector replaces the vector with which it was compared; otherwise, the old vector is retained. This basic idea is extended by perturbing an existing vector through the addition of one or more weighted difference vectors to it.

The perturbation scheme generate a variation v according to the following expression,

$$v = x_i^k + L(x_b^k - x_i^k) + F(x_{r_2}^k - x_{r_3}^k) \quad (1)$$

where x_i^k is the parameter vector to be perturbed at iteration k , x_b^k is the best parameter vector of the population at iteration k , $x_{r_2}^k$ and $x_{r_3}^k$ are parameter vectors chosen randomly from the population and are different from running index i . L and F are real and constant factors which controls the amplification of the differential variations $(x_b^k - x_i^k)$ and $(x_{r_2}^k - x_{r_3}^k)$. This expression has two different terms: the $(x_b^k - x_i^k)$ term is a kind of stochastic gradient while $(x_{r_2}^k - x_{r_3}^k)$ is a kind of random search.

In order to increase the diversity of the new generation of parameter vectors, crossover is introduced. Denote by $u_i^k = (u_{i,1}^k, u_{i,2}^k, \dots, u_{i,D}^k)^T$ the new parameter vector with

$$u_{i,j}^k = \begin{cases} v_{i,j}^k & \text{if } p_{i,j}^k < \delta \\ x_{i,j}^k & \text{otherwise} \end{cases} \quad (2)$$

where $p_{i,j}^k$ is a randomly chosen value from the interval $[0, 1]$ for each parameter j of the population member i at step k and δ is the crossover probability and constitutes the crossover control variable. The random values $p_{i,j}^k$ are made anew for each trial vector i .

To decide whether or not vector u_i^k should become a member of generation $i + 1$, the new vector is compared to x_i^k . If vector u_i^k yields a better value for the objective fitness function than x_i^k , then is replaced by u_i^{k+1} ; otherwise, the old value x_i^k is retained for the new generation. The general idea of the previous mechanism: mutation, crossover and selection are well known and can be found in literature (Goldberg 1989).

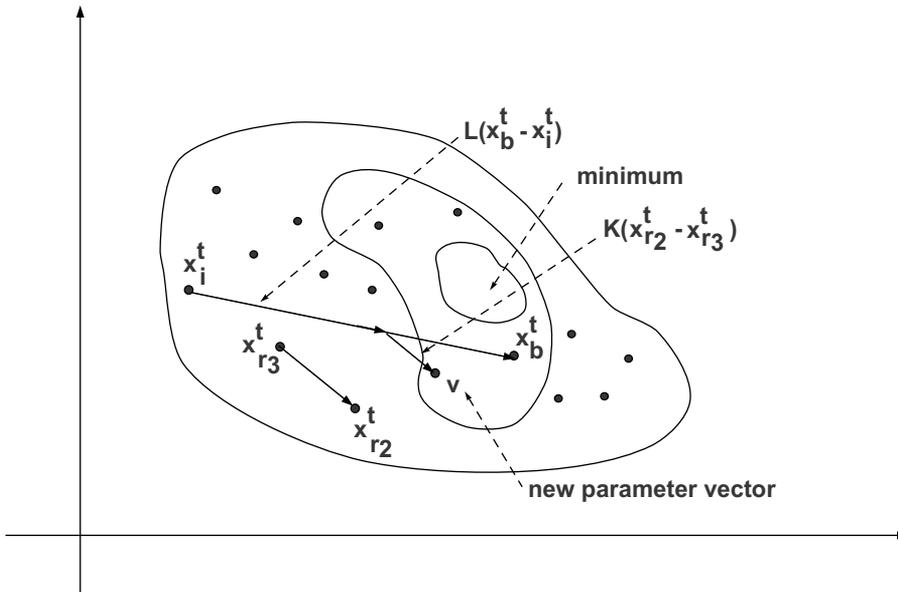


Figure 1. New population member generation

A. Fitness function

Due to we are trying to localize a mobile robot, the natural choice for fitness function is the sum of squared errors function. If the observation vector at time t is $z_t = (z_{1,t}, \dots, z_{p,t})^T$ and the predicted observations according the estimated robots pose is $\hat{z}_t = (\hat{z}_{1,t}, \dots, \hat{z}_{p,t})^T$ then the penalty function can be stated as

$$L(x_t - \hat{x}_t) = (z_t - \hat{z}_t)^T (z_t - \hat{z}_t) \quad (3)$$

In the global localization problem there exist some aspects that make this fitness function difficult to manage:

- The range and accuracy of the sensor limits the possibility of discriminate between different poses, leading the fitness function to a high number of global maxima.
- The number of sensors limits the possibility of discriminate between robot's poses leading to multiple global maxima in the fitness function.

- The geometrical similarities in the environment due to the repetition of the space distribution originate the presence of a high number of possible robot's pose solutions to the mean square loss function.

The loss function defined in this way provides us with an optimization mechanism which obtain an unstable parameter solution and by extension an unstable filter. The instability of an evolutive filter based on this penalty function derives from the environment ambiguities. Due to that the evolutive filter can move from one local minimum to other, originating abrupt changes in the pose estimate. This problem comes from the fact we are not using all information we know about the system at the loss function. In fact, we only use the observation model to predict the sensor measurements at each position, and these predicted measurements together with the actual measurements are introduced in the loss function to evaluate the robot's pose estimate. The instability originated by the existence of multiple solutions to the loss function, can not be solved by considering only the available sensor observations. One possibility consists of introducing in the loss function all the information available about the system.

The solution proposed introduced all available information in two ways:

- *State space model information.*

State transition model and observation model are used to estimate the robot's pose and robot's observations next cycle. If we define $x_{i,t}$ as a possible robot's pose solution i obtained in the last iteration of the algorithm at step t . According to the robot motion model the estimated position at step $t + 1$ according to the odometric information available will be

$$x_{i,t+1} = g(x_{i,t}, u_t) \quad (4)$$

This information is used for moving the poses of all population members $\{x_i^m\}_t$ obtained in the last iteration m of the evolutive filter at step t . A faster approach, from a computational point of view, consist of estimating only the movement of the best estimate $\hat{x}_{t+1} = g(\hat{x}_t, u_t)$ to estimate a displacement $\hat{d}_t = \hat{x}_{t+1} - \hat{x}_t$ and then to apply to all population member. The initial population at step $t + 1$ will be

$$\{x_i^0\}_{t+1} = \{x_i^m\}_t + \hat{d}_t \quad (5)$$

In a similar form the observation model together with the state estimate let us predict the sensor observation values $\hat{z}_t = h(\hat{x}_t)$.

- *Probabilistic information.*

To eliminate the ambiguity, the statistical information available have to be considered. This information is included into the loss function. One possible choice is to include the distance between the estimated robot's pose \hat{x}_{t+1} and each candidate solution x_i^k at iteration k at step t ; in the loss function together with the observation error between sensor prediction and sensor measurement.

$$L(\hat{x}_{t+1} - x_i^k) = (\hat{z}_t - z_t)^T Q^{-1} (\hat{z}_t - z_t) + [(\hat{x}_{t+1} - x_i^k)^T R^{-1} (\hat{x}_{t+1} - x_i^k)^T] \quad (6)$$

where Q and R are the covariance matrix of the observation and motion noises, and Z the sensor data. This loss function includes two effects: the first term considers the observational error, that is the squared distances of sensor observation innovation, and the second term considers the motion innovation. The motion innovation considers the distance between the predicted robot position and the position for a particular solution weighted according a coefficient.

B. The Evolutive Localization Filter algorithm

According with the previous ideas algorithm has the following steps:

- Step 1: Initialization.

The initial set of solutions is calculated and the fitness value associated to each of the points in the state space is evaluated. In the most general case where no information about initial position is available, the initial set of robot's pose solutions is obtained by drawing the robot's poses according to a uniform probability distribution over the state space.

The initial robot's pose estimate is fixed to an initial value.

- Step 2: Evolutive search.

(a) For each element of the set of robot's pose solutions, and according to the map, the expected sensor observations are obtained. The expected observation, the sensor observations, the robot's pose estimate and the robot's pose element are used to evaluate the loss function for each pose element in the set of solutions.

(b) A new generation of perturbed robot's pose solutions are generated according to the perturbation method exposed previously. For each perturbed solution the expected observations are calculated and the loss function evaluated. If the perturbed solution results in a better loss function, this perturbed solution is selected for the following iteration, otherwise the original is maintained.

(c) The crossover operator is applied to the resultant population.

(d) The robot's pose element of the set with lower value of the loss function is marked as best robot's pose estimate. Go to step 2b a given number of iterations.

- Step 3: Updating. The best robot's pose element of the population is used as the updated state estimate and then used in state transition model to predict the new state according to the odometry information.

$$\hat{x}_{t+1} = f(\hat{x}_t, u_t) \quad (7)$$

Then, the displacement is evaluated and the whole population is moved according to this displacement. Then go to step 2.

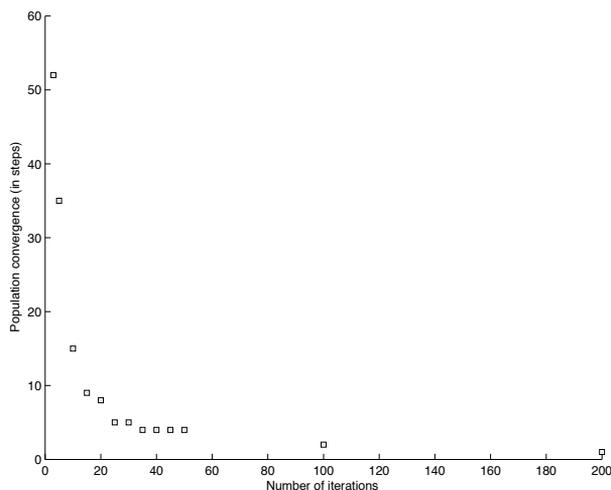


Figure 2. Convergence speed as a function of the iteration number

3. Experimental results

All experiments have been developed in an indoor environment - University laboratories, offices and corridors, and conducted on a B21-RWI mobile vehicle, equipped with laser range finder.

A. Convergence of the algorithm

One fundamental question about evolutionary algorithms is their convergence and their convergence rates: how quickly can the populations come to the individuals (robot's pose solutions) with the highest fitness value?

It has been considered that the algorithm converges to a solution, when all population size is located in a ball of given radius around the best estimate (0.5 meters). Fig. 2 shows the convergence speed of the Evolutive Filter with a set of 500 individuals as a function of the iteration number used for a given trajectory. . As figure 2 suggest, the algorithm proposed is highly efficient in terms of convergence speed. The algorithm behaviour is clearly asymptotical. The speed of convergence grows with the iteration number, but an excessively fast convergence can lead to a local minima. In the tests, the appropriate range of iterations is located between 10 and 30. In this range, the probability of converging to a local minimum is low and the computational effort is reasonable.

Fig. 2 shows a typical convergence process of the Evolutive Localization Filter for a population set of 1000 individuals and 15 iterations of the evolutive filter in each cycle. The robot trajectory starts in the left office of the upper side of the map, goes to the corridor, and continue along the central corridor toward the right side of the map (Fig. 2). It can be noticed that the population individuals tend to concentrate in highly similar areas at the initial pose. After some cycles the algorithm estimate (black circle) reaches the true robot's pose (blue cross) and the population set converges to the true robot's pose. The environment is modelled as an occupancy grid with a cell's size of 12 centimetres. The map scales in figure 2 are expressed in cells.

B. Accuracy and robustness

One striking aspect is the low number of robot's pose solutions required in the population. An appropriate population size for the environment under consideration is 300. The size of

this environment is approximately $865 m^2$, and then less than 0.3 elements per square meter is required. In a similar test with Monte Carlo, for this environment are required 10000 samples to localize reliably the mobile robot. Jensfelt tests with a Monte Carlo Localization method (Jensfelt, 2001) for a test environment of $900m^2$ required 10,000 samples. That is 11 samples per square meter.

Fig. 3 shows the performance of the ELF under different sensor noise levels. The ELF algorithm exhibit an relatively constant average error around 6 cm until a sensor noise level of 30%, but even with higher noise levels (50%) is able to localize the robot. This robustness to sensor noise is equivalent to Monte Carlo results shown in (Thrun & Fox, 2001).



Figure 3. EFL convergence process at cycles: 3, 5, 7 and 9

If we consider the effect of the population size on the accuracy of the algorithm, it can be noticed the minimum effect of the population size on the accuracy. This behaviour is consequence of the search nature of the ELF algorithm. This behaviour differs completely from Monte Carlo results. As noticed by several authors (Doucet, 1998, Liu & Chen, 1998), the basic Monte Carlo filter performs poorly if the proposal distribution, which is used to generate samples, place not enough samples in regions where the desired posterior is large. This problem has practical importance because of time limitations existing in on-line applications.

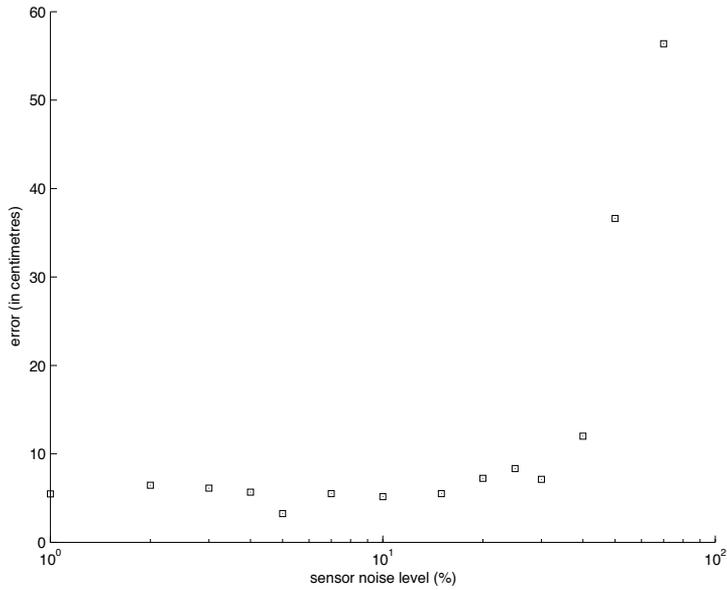


Figure 4. Error of ELF as a function of the sensor noise

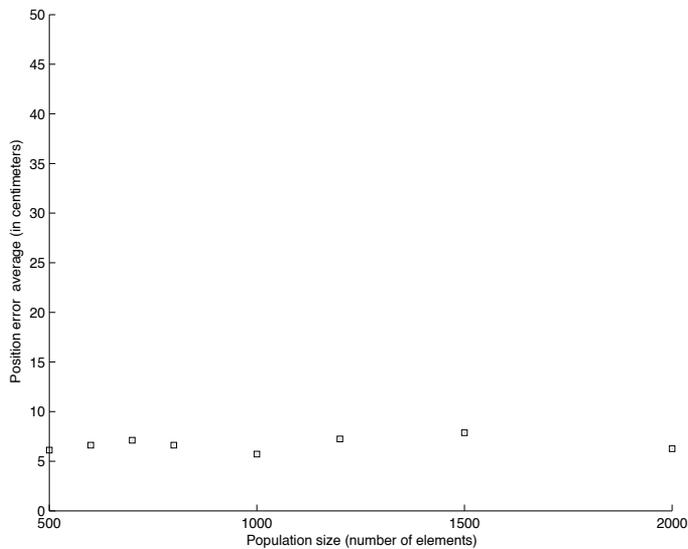


Figure 5. Accuracy as a function of the population size

C. Complexity and computational cost

A final issue addressed in our experiments concerns the running time of ELF algorithm. The absolute time depends on several factors: the computer platform, the observation prediction model and the sensor data, and the population and iterations number. This section illustrates the requirements of ELF. All results reported here were obtained in a PC with an Athlon XP 1800+ processor.

Table 1 shows the time required for ELF to update the estimate value using 60 range laser data, 15 iterations and different population sizes. Table 2 shows the time required for ELF algorithm to update the robot's pose estimate using a fixed population size and changing the iteration's number. In both cases the behaviour of the algorithm is completely linear. In our experimental localization tests, in a test area of 865m² the best results have been obtained with an initial population of 300 elements and 15 iterations. Once the population converge into a ball of 1 meter radius, the population required to keep the robot localized decrease considerably. In our experiments 50 elements are sufficient. This results shows that even in the initial steps of the ELF algorithm the computational cost is moderate.

Population/Iterations	Time(ms)
1000/15	1703
500/15	859
400/15	704
300/15	520
200/15	359
100/15	180
75/15	141
50/15	94

Table 1. Computation time for a fixed number of iterations

Population/Iter.	Time(ms)
500/5	328
500/10	641
500/15	859
500/20	1125
500/25	1407

Table 2. Computation time for a fixed population of 500 elements

A critical point in any global localization algorithm is the variation in the computational requirements with the environment dimensions. EFL algorithm requirements are proportional to the surface of the environment map. In the experiments done in our laboratory test site, to cope a 900m² area the algorithms requires 300 samples and for an area of 1800m² the samples required are 600. After the population convergence, the samples can be reduced to 30 elements.

4. Conclusions

The proposed algorithm has a range of interesting characteristics:

Evolutionary filters can deal with non-linear state space dynamics and noise distributions.

Due to the set of solutions does not try to approximate posterior density distributions, it does not requires any assumptions on the shape of the posterior density as parametric approaches do.

Evolutionary filter focus computational resources in the most relevant areas, by addressing the set of solutions to the most interesting areas according to the fitness function obtained.

The number of tentative solutions required in the evolving set is much lower than those required in particle filters, and similarly to those filters the evolving set can be reduced when the algorithm has converged to a reduced area around the best estimate.

The size of the minimum solution's set required to guaranty the convergence of the evolutionary filter to the true solution is low.

Due to the stochastic nature of the algorithm search of the best robot's pose estimate the algorithm is able to cope a high level of sensor noise with low degradation of the estimation results.

The algorithm is easy to implement, and the computational cost makes it able to operate on line even in relatively big areas.

The proposed method has been tested under real conditions in a B-21 mobile robot.

Acknowledgment

This research is sponsored by Spanish Government (MICYT) under agreement number DPI2003-01170.

5. References

- Arras, K., Castellanos, J.A., and Siegwart, R. (2002). Feature-based multi-hypothesis localization and tracking for mobile robots using geometric constraints. *Proc. of the Int. Conference on Robotics and Automation ICRA-02*, Washington D.C., USA, pp 1371-1377.
- Austin, D.J., and Jensfelt, P. (2000). Using multiple Gaussian hypotheses to represent probability distributions for mobile robot localization. *Proc. of the Int. Conference on Robotics and Automation ICRA-00*, San Francisco, CA, USA, pp 1036-1041.
- Burgard, W., Fox, D., Henning, D., and Schmidt, T. (1996). Estimating the absolute position of a mobile robot using position probability grids. *Proc. of the National Conference on Artificial Intelligence AAAI-96*, Portland, Oregon, USA, pp 896-901.
- Cox, I.J., Leonard, J.J. (1994). *Modelling a dynamic environment using a Bayesian multi hypothesis approach*. *Artificial Intelligence*, 66, pp 311-44.
- Dellaert, F., Fox, D., Burgard, W., Thrun, S. (1999). Monte Carlo Localization for Mobile Robots. *Proceedings of the 1999 International Conference on Robotics and Automation*, pp. 1322-1328 .
- Doucet, A. (1998). On sequential simulation-based methods for Bayesian filtering. *Technical Report CUED/FINFENG/TR 31*, Cambridge University, Dept. of Engineering, Cambridge, UK.
- Fox, D., Burgard, W., and Thrun, S. (1999). *Markov localization for mobile robots in dynamic environments*. *Journal of Artificial Intelligence Research*, 11, pp 391-427, 1999.

- Goldberg, D.E. (1989) *Genetic algorithm in Search, Optimization, and Machine Learning*. Addison Wesley Publishing Company.
- Jensfelt, P., Kristensen, S. (1999). Active global localisation for a mobile robot using multiple hypothesis tracking. *Proc. IJCAI Workshop on Reasoning with Uncertainty in Robot Navigation*, pp 13-22, Stockholm, Sweden.
- Jensfelt, P., Wijk, O., Austin, D.J. and Andersson, M. (2000). Experiments on augmenting condensation for mobile robot localization. *Proc. of the Int. Conference on Robotics and Automation ICRA-00*, San Francisco, CA, USA, pp 2518-2524.
- Jensfelt, P. (2001). *Approaches to mobile robot localization in indoor environments*. Doctoral Thesis, Royal Institute of Technology, Sweden.
- Liu, J., Chen, R. (1998). Sequential Monte Carlo methods for dynamic systems. *Journal Amer. Statis. Assoc.*, 93, pp 1032-1044.
- Reuter, J. (2000). Mobile robot self-localization using PDAB. *Proc. IEEE International Conference on Robotics and Automation (ICRA-2000)*, San Francisco, C.A.
- Roumeliotis, S.I., Bekey, G.A. (2000). Bayesian estimation and Kalman filtering: A unified framework for mobile robot localization, in *Proc. IEEE International Conference on Robotics and Automation (ICRA-2000)*, San Francisco,, pp 2985-2992.
- Storn, R. and Price, K. (1995). Differential Evolution- A simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-01.
- Thrun, S., Fox, D., Burgard, W. and Dellaert, F. (2001). *Robust Monte Carlo localization for mobile robots*. *Artificial Intelligence*, 128, pp 99-141.

Learning to Walk with Model Assisted Evolution Strategies

Matthias Hebbel and Walter Nisticò
Technische Universität Dortmund
Germany

1. Introduction

Many algorithms in robotics contain parameterized models. The setting of the parameters in general has a strong impact on the quality of the model. Finding a parameter set which optimizes the quality of the model typically is a challenging task, especially if the structure of the problem is unknown and can not be specified mathematically, i.e. the only way to get the function value for a certain parameter set is to test them on the real problem. Controlling the robot's legs in order to walk is an example of such kind of *black box* approach. The walk is usually defined by three-dimensional trajectories for the movement of the legs and feet. The parameters (e.g. step height, step length, timings etc.) of these trajectories affect the stability and speed of the walking gait. Finding a parameter set which creates an optimal (in this case fast) walk can not be done by hand, since the direct effect of a parameter change is unclear. This chapter describes how Evolution Strategies can be used for the gait optimization of a four-legged robot whose walk is defined by 31 parameters and presents how Model Assisted Evolution Strategies lead to a faster convergence of the optimization of the problem.

At first an introduction to the standard Evolution Strategy with self-adaptation of the mutation strengths is given. The individuals of the Evolution Strategy here define a set of walk parameters. To evaluate the fitness of each individual, the robot has to walk for a certain time with this parameter setting and measure the resulting walk speed. This is not only time consuming but also causes a lot of wear out of the robot – especially parameters resulting in stumbling or tumbling can damage the robot. Consequently, the goal is to minimize the amount of real fitness evaluations on the robot. For that reason Model Assisted Evolution Strategies will be introduced. These strategies use the fitness measurements as obtained samples from the search space to predict the fitness of the individuals and – based on the model – sort out the least promising individuals, resulting in a faster convergence of the optimization process.

Since the quality of the model used for fitness prediction has a big effect on the convergence rate of the Model Assisted Evolution Strategy, mechanisms for controlling the impact of the model will be explained. The chapter closes with the results and comparisons of the walk learning process of the standard Evolution Strategy and its model assisted modifications.

2. The Standard Evolution Strategy

The Evolution Strategy (ES) has been developed by Schwefel (Schwefel, 1975) and Rechenberg (Rechenberg, 1973) for the optimization of technical problems. It is inspired by the biological evolution and is working with a population of individuals. Each individual represents a point \vec{x} in the real-valued search space; the point \vec{x} is encoded in the object parameters x_1, \dots, x_n of the individual and represents its genes. In terms of Evolution Strategies each individual can be assigned a function value $f(\vec{x})$ (in this case scalar) representing the quality of the solution, the so called fitness which has to be optimized. The process of the evolution is shown in Fig. 1. An amount of μ individuals constitute the parent population and create a new offspring population of the next generation with λ individuals. An offspring individual is created by recombining the real-valued genes of ρ randomly selected parents followed by a random mutation of each object parameter such that the offsprings differ in their genes from their parents. The fitness of each individual in the offspring generation will be measured and the selection operation chooses the μ best individuals to establish the new parent generation.

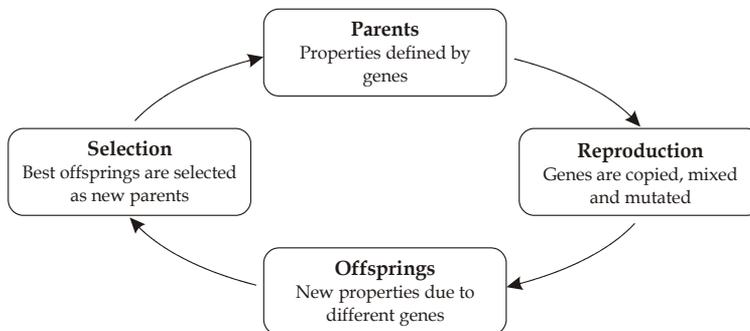


Figure 1. Optimization cycle of the standard Evolution Strategy

Offspring individuals are generated by means of recombination from the parent individuals. They can be recombined in a discrete or intermediate way. In case of the discrete recombination each object parameter x_i is selected randomly from the according object parameter p_i from one out of the ρ parents. The intermediate recombination on the other hand creates the components x_i out of the mean value of all p_i of the ρ parents. In both recombination variants the ρ parents are chosen randomly from the parent population, their fitness does not affect the probability to be chosen.

The leading part in the genetic modification of individuals is played by the mutation operator. The mutation operator is executed after the recombination. Each object parameter x_i is modified by adding a normally distributed random value $N(0, \sigma_i^2)$ with expected value 0 and standard deviation σ_i :

$$\tilde{x}_i = x_i + N(0, \sigma_i^2). \quad (1)$$

The standard deviation conforms to the mutation strength: the bigger the standard deviation, the higher is the probability of a greater mutation and vice versa. Each object parameter x_i has its own mutation strength σ_i . The values of the mutation strengths σ_i have a big effect on the progress speed of the evolution process. With a mutation strength chosen

too big, the chance to overshoot the optimum rises, while small mutation strengths increase the probability to get stuck in a local optimum and furthermore the speed to approach the optimum is unnecessary small. Hence it is very important to adapt the mutations strengths to the current situation. For that reason Rechenberg developed the 1/5 rule for an Evolution Strategy with 1 parent and 1 offspring. He proved for two models that the mutation strength is optimal if 1 out of 5 generated offsprings is fitter than the previous parent (Rechenberg, 1973). In case of a bigger success rate the mutation strength is too small and has to be increased, otherwise it is too big and has to be decreased.

Schwefel improved the control of the mutation strengths and extended it to Evolution Strategies with more than 1 offspring (Schwefel, 1975). Each object parameter carries its own mutation strength (strategy parameter). These strategy parameters are also included in each individual and are selected and inherited together with the individual's assignments for the object parameters. Thus, they have a higher probability to survive when they encode object parameter variations that produce fitter individuals. This adaptation of the mutation strength is called **self-adaptation** (Beyer, 1996). The recombination of the strategy parameters can be discrete or intermediate. The mutation of the strategy parameters is executed before the mutation of the object parameters with

$$\tilde{\sigma}_i = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1)). \quad (2)$$

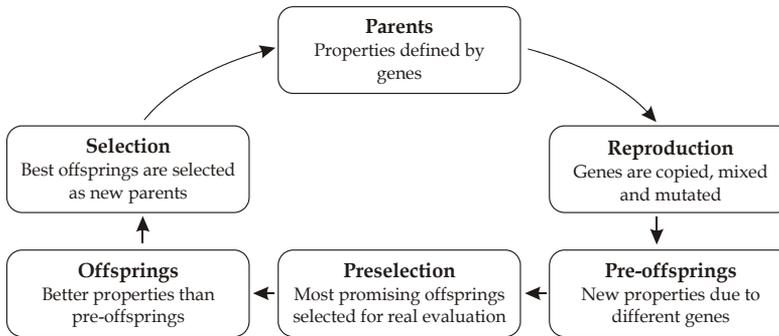
The mutation consists of the random number $\tau' \cdot N(0,1)$ which is equal for all variances σ_i of the individual while the random number $\tau \cdot N_i(0,1)$ is varied for each σ_i . Schwefel recommends $\tau' \sim 1/\sqrt{2n}$ and $\tau \sim 1/\sqrt{2\sqrt{n}}$ for an n -dimensional search space.

After the fitness values of the offspring individuals have been assigned, the selection operator selects the parents for the next generation. In case of the so called "+" strategy it chooses the μ best individuals out of the offspring **and** the parent population, while in case of the "," strategy only the best μ individuals out of the offspring population are selected to become the parents of the next generation. Additionally the lifetime of an individual can be limited to κ generations in case of the + selection, creating a mixture of the + and , selection. The evolution loop can be stopped after a satisfying solution has been found, a number of generations have passed or the fitness does not improve over a number of generations, indicating a convergence.

3. Model Assisted Evolution Strategies

The basic idea of a Model Assisted Evolution Strategy (MAES) is to generate a higher amount of offsprings λ_p with $\lambda_p > \lambda$ and to preselect the most promising candidates. Out of the λ_p offsprings the λ best ones will be preselected based on their predicted fitness. The fitness is predicted from a model which takes the previously measured fitness values into account. The λ preselected individuals will afterwards be evaluated and the μ best chosen as parents for the next generation like in the standard Evolution Strategy, Fig. 2 visualizes this extension to the standard Evolution Strategy. Notice that the fitness of the offsprings which are available to become parents is always evaluated on the real problem and therefore the individuals are always assigned the true fitness. This way a convergence to a wrong optimum caused by wrong fitness predictions by the model is excluded.

If the model manages to sort out the worst candidates and to preselect the best ones, the convergence of the Evolution Strategy is accelerated, even though the number of real fitness



evaluations is the same as in the case of the standard Evolution Strategy. The following section will explain the three models used in this work for fitness approximation.

Figure 2. Evolution loop of a Model Assisted Evolution Strategy

3.1 Modelling the Search Space

During the evolution process the fitness measurements give an indication for the structure of the search space, they represent known scalar function values. Based on these known function values the model has to be able to make predictions of function values of unobserved points. Mathematically speaking the model must approximate the function $f(\vec{x}): \mathbb{R}^n \rightarrow \mathbb{R}$ based on d previously observed data points which are stored (with the according measured fitness values) in the data base D for any query point $\vec{q} \notin D$. The estimated function value will be called $\hat{f}(\vec{q})$ and should be close to the real (unknown) function value $f(\vec{q})$.

Principally the model can be global or local. Global models are valid for the whole search space and can be used for estimation of any query point \vec{q} . They have only to be calculated when new points are integrated in the underlying data base D , i.e. for Evolution Strategies after each generation. The biggest problem for global models is the typically small number of points in D relative to the dimension of the usually multimodal search space, leading to a poor general approximation. Local models on the other hand are only valid for a small region around a query point \vec{q} and have to be recalculated for each approximation of $\hat{f}(\vec{q})$. Local models normally only take a subset of points in D closest to \vec{q} into account since these points carry the most information about $\hat{f}(\vec{q})$. Notice that the data base D grows during the evolution process: each individual whose true fitness is evaluated is added with its measured fitness value.

The simplest model is the **Nearest Neighbour** approximation. It estimates the function value of a query point \vec{q} with the function value of its nearest neighbour among the previously measured points in the data base D . The distance Δ between two points \vec{x} and \vec{q} can be the Euclidian distance

$$\Delta(\vec{x}, \vec{q}) = \sqrt{\sum_{i=1}^n (x_i - q_i)^2} \quad (3)$$

which is appropriate if each dimension has the same scaling and a comparable gradient. If the dimensions have different scales, they can be standardized to have a standard deviation of 1 to make them comparable. For distance calculation each dimension i then has to be scaled by the reciprocal of the standard deviation $\sigma_i = \sqrt{\text{var}(x_i^{(l)})}$, $1 \leq l \leq d$ with the weight $w_i = 1/\sigma_i$

$$\Delta(\bar{x}, \bar{q}) = \sqrt{\sum_{i=1}^n (w_i (x_i - q_i))^2}. \quad (4)$$

In case of this Nearest Neighbour model the approximated function value $\hat{f}(\bar{q})$ is given by

$$\hat{f}(\bar{q}) = f(\bar{x}_{\min}), \quad \bar{x}_{\min} = \arg \min_{l=1, \dots, d} \left\{ \Delta(\bar{x}^{(l)}, \bar{q}) \right\} \quad (5)$$

This model is computationally very efficient; however it has the disadvantage that it does not interpolate between measured points which could lead to the problem that several offsprings will be assigned the same fitness value and the preselection operation can not distinguish their quality.

The **polynomial model** uses linear regression to fit the measured points in D to a polynomial function to overcome this problem. At first the standard linear regression will be explained and then extended to the weighted linear regression.

An n -dimensional polynomial of second-order is specified by

$$\hat{f}(\bar{x}) = a_0 + \sum_{1 \leq i \leq n} a_i x_i + \sum_{1 \leq i \leq j \leq n} a_{n-1+i+j} x_i x_j \quad (6)$$

with the coefficients a_i . The problem in regression is now to determine the coefficients a_i such that $\hat{f}(\bar{x})$ approximates $f(\bar{x})$ in all d points of the data set D as good as possible. In mathematical notation the squared error function

$$\varepsilon(\bar{a}) = \sum_{i=1}^d \left(f(\bar{x}^{(i)}) - \hat{f}(\bar{x}^{(i)}) \right)^2 \quad (7)$$

has to be minimized. Written as a matrix equation with the measured function values $y^{(1)}, \dots, y^{(d)}$ in D

$$\bar{y} = \left[y^{(1)}, \dots, y^{(d)} \right]^T \quad (8)$$

and the matrix X representing the second-order polynomial

$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & \left(x_n^{(1)}\right)^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(d)} & x_2^{(d)} & \dots & \left(x_n^{(d)}\right)^2 \end{bmatrix} \quad (9)$$

the equation

$$\bar{y} = X \cdot \bar{a} \quad (10)$$

holds for an exact approximation.

If the polynomial can not approximate the function values exactly, an error term \vec{v} has to be added:

$$\vec{y} = \mathbf{X} \cdot \vec{a} + \vec{v}. \quad (11)$$

As mentioned before, the error term $\varepsilon(\vec{a})$ has to be minimized and equation (7) can be written in matrix notation as

$$\varepsilon(\vec{a}) = \sum_{i=1}^d \left(f(\vec{x}^{(i)}) - \hat{f}(\vec{x}^{(i)}) \right)^2 = \vec{v}^T \cdot \vec{v} = (\vec{y} - \mathbf{X} \cdot \vec{a})^T (\vec{y} - \mathbf{X} \cdot \vec{a}). \quad (12)$$

For the minimization of $\varepsilon(\vec{a})$ it is sufficient to postulate that $\nabla(\varepsilon(\vec{a})) = 0$, resulting in the solution for the coefficients \vec{a}

$$\vec{a} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \vec{y}. \quad (13)$$

This regression method can be used to build a global model if all points in D are used to estimate the coefficients \vec{a} . The disadvantage is that an underlying multimodal search space requires a polynomial of high order to be approximated closely. Unfortunately the number of coefficients rises disproportionately with the order of the polynomial; a polynomial of second order already has $(n+1)(n+2)/2$ coefficients (Jin, 2003) and still can only approximate the function with a unimodal polynomial. This least square method requires at least as many samples in the data base D as coefficients exist.

This problem can be overcome with a local model. The model becomes local if the distances to the query point are weighted and represent the fact that the close points carry more information about the query point \vec{q} than points which are further away. All points in D can be taken or, since far points would get a weight close to 0, only a subset of the k closest points. The weight w_l is a function of the distance between the query point \vec{q} and the l -th measured point $\vec{x}^{(l)}$ in the data base. As suggested in (Hoffmann & Hölemann, 2006) the weight is determined by a Gaussian kernel function

$$w_l \left(\Delta(\vec{x}^{(l)}, \vec{q}) \right) = \exp \left(-\Delta(\vec{x}^{(l)}, \vec{q})^2 \right). \quad (14)$$

The distance is measured as suggested in equation (4).

For the local model a polynomial of first order will be used which is given for an n -dimensional space by

$$\hat{f}(\vec{x}) = a_0 + \sum_{l=1}^n a_l x_l \quad (15)$$

with the coefficients a_0, \dots, a_{n+1} . The weights for the k points are stored in the diagonal matrix

$$\mathbf{W} = \begin{pmatrix} w_1 & 0 & \dots & 0 \\ 0 & w_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & w_k \end{pmatrix} \quad (16)$$

and the error function (12) results in the weighted case to

$$\varepsilon(\bar{a}) = (\bar{y} - X \cdot \bar{a})^T W (\bar{y} - X \cdot \bar{a}). \quad (17)$$

The optimal coefficients then can be calculated with

$$\bar{a} = (X^T W X)^{-1} X^T W \bar{y}. \quad (18)$$

Another used model is the approximation with a **Gaussian Process**. The Gaussian Process is a probabilistic model which can calculate besides the predicted value also a confidence value representing the certainty of the prediction. In this work the implementation of Rasmussen with automatic relevance determination has been used. Explaining prediction with a Gaussian Process in detail would go beyond the scope of this chapter, very good detailed descriptions of Gaussian Processes are given in (Gibbs & MacKay, 1997; Rasmussen & Williams, 2006).

3.2 Controlling the Model Impact

Theoretically the Model Assisted Evolution Strategy already converges faster than the standard Evolution Strategy when the model performs better than a random preselection of the individuals would perform (Ulmer et al., 2004). However for a model which performs worse than a random selection, the progress speed can be reduced or optima might even not be reached because the individuals leading towards the optimum always get sorted out by the preselection. For that reason it makes sense to control the impact of the model based on the capability of the model to preselect the individuals.

For the preselection process the capability of the model to select the most promising individuals is of importance, while the ability to estimate the exact fitness value is of little interest. Thus the quality of the model is defined according to (Ulmer et al., 2004; Jin et al., 2003), as follows: Since only the real fitness values of the λ offsprings are known, the quality will be evaluated based on the comparison of the selected μ individuals based on the real fitness measurement and the best μ individuals that would have been selected based on the predicted fitness. The λ offsprings are sorted according to their true fitness and a weight $(\lambda - i)$ with i representing their position is assigned to each individual. Thus the individual with the best fitness would get a weight $(\lambda - 1)$, the second best $(\lambda - 2)$ etc. Now the weights for the individuals that would have also been selected by the fitness estimated by the model are summed up:

$$Q = \sum_{i=1}^{\mu} g_i (\lambda - i) \quad (19)$$

with $g_i = 1$ if the individual would have been selected and $g_i = 0$ otherwise. This quality measure takes the true rank of the individuals into account and reflects the fact that a wrongly classified best individual is punished more than for example a missed second best. The maximum quality as a result is given by

$$Q^{max} = \sum_{i=1}^{\mu} (\lambda - i) = \mu\lambda - \frac{\mu(\mu + 1)}{2} \quad (20)$$

and the quality of a random preselection is defined by

$$Q^{rand} = \sum_{i=1}^{\mu} i \frac{\binom{\mu}{i} \binom{\lambda-\mu}{\mu-i}}{\binom{\lambda}{\mu}} \cdot \frac{1}{\mu} \sum_{i=1}^{\mu} (\lambda-i) = \frac{\mu^2}{\lambda} \frac{2\lambda - \mu - 1}{2}. \quad (21)$$

Two approaches based on the current quality of the preselection to influence the impact of the model will be used in this work.

The **Controlled Model Assisted Evolution Strategy** (C-MAES) adapts the influence of the model by varying the amount of offsprings λ_p which will be evaluated by the model. A good quality of the model results in a bigger trust in the model and λ_p can be increased, while for a poor model quality λ_p has to be decreased. After each generation t the selection quality $Q^{(t)}$ gets re-evaluated and $\lambda_p^{(t+1)}$ adapted according to the control law suggested by (Ulmer et al., 2004). If the quality is better than the random quality, the amount of offsprings is increased by

$$\lambda_p^{(t+1)} = \lambda_p^{(t)} + \frac{Q^{max} - Q^{(t)}}{Q^{max} - Q^{rand}} \cdot \delta_{\lambda_p}, \quad (22)$$

if the quality on the other hand is worse than the random quality, the model impact is decreased by

$$\lambda_p^{(t+1)} = \lambda_p^{(t)} - \frac{Q^{max} - Q^{(t)}}{Q^{rand}} \cdot \delta_{\lambda_p}. \quad (23)$$

The factor δ_{λ_p} denotes the adaptation rate.

Hoffmann and Hölemann suggest the adaptation of the size of the offspring generation λ in order to have a better control over the model impact (Hoffmann & Hölemann, 2006). This variant is called **λ -Controlled Model Assisted Evolution Strategy** (λ -CMAES). Here the amount of offsprings λ_p that is undergoing the preselection is kept constant. The idea is that in case of a good quality of the preselection only a smaller amount of offsprings λ has to be evaluated on the real problem, since in case of a good model already the best individuals from the λ_p have been preselected. The amount of offsprings $\lambda^{(t+1)}$ for the next generation $t+1$ is updated according to the following control laws: if the current model quality is better than a random model, i.e. $Q^{(t)} > Q^{rand}$, the $\lambda^{(t+1)}$ is decreased, giving more trust in the model by

$$\lambda^{(t+1)} = \max \left(\lambda^{(t)} - \frac{Q^{rand} - Q^{(t)}}{Q^{rand}} \cdot \delta_{\lambda}, \mu \right). \quad (24)$$

If the model actual quality is worse than the quality of the random model, the trust in the model is decreased by increasing $\lambda^{(t+1)}$ according to

$$\lambda^{(t+1)} = \min \left(\lambda^{(t)} + \frac{Q^{max} - Q^{(t)}}{Q^{max} - Q^{rand}} \cdot \delta_{\lambda}, \lambda_p \right). \quad (25)$$

A particularity to be mentioned is that the previously explained quality calculation becomes unstable for $\lambda = \mu$ since all individuals would always be selected correctly. To ensure a sufficient statistical basis, μ used in the quality calculation is replaced by $\tilde{\mu} = \lambda/2$ if λ drops below 2μ (Hoffmann & Hölemann, 2006).

4. Learning to Walk with Evolution Strategies

The performance of the different Evolution Strategies will be evaluated for a robot learning to walk. The problem of learning to walk is reduced to optimizing parameters for a given parameterized model which generates walk movements for a robot.

The used robot platform for the presented research is a simulated model of the formerly commercially available Sony Aibo ERS-7 shown in Fig. 3. The Aibo is a quadruped robot with three degrees of freedom in each leg, i.e. a hip abduction, a hip flexion and a knee flexion joint. SimRobot (Laue et al., 2006) has been used as the simulator. It is a general robot simulator using the OpenDynamicsEngine for the physical simulation. The physical properties of the simulated robot like maximum force and maximum speed of the motors in the joints have been specially trained to behave like the joints of the real robot.



Figure 3. Sony Aibo robot ERS-7

The most significant achievement concerning the walk on this robotic platform has been presented by Hengst et al. with the so-called *PWalk* (Hengst et al., 2002) which achieves a very good stability using a particular posture of the front legs, where the contact point with the ground is located on the forearm instead of the paw. As shown in Fig. 4., this approach moves the robot's legs on predefined loci. The required joint angles for the movement of the legs are then calculated using inverse kinematics. Following a trot gait, the two diagonal opposite legs are always moved at the same time, i.e. two legs are in the air while the other two legs remain in contact with the ground. The loci defining the foot movement are represented in a parameterized form (Hebbel et al., 2006). Each locus is defined by four 3-dimensional points P_1, \dots, P_4 both for the front and hind legs (see Fig. 4.). The time parameter t_{step} defines the total time for a complete step. Additionally 3 timing parameters t_1, t_2, t_3 are used for each locus to specify the relative amount of time needed for the foot to travel from vertex P_i to P_{i+1} for $1 \leq i < 4$. These relative times sum up to 1, consequently the time t_4 to travel from P_4 to P_0 is defined by $t_4 = 1 - t_1 - t_2 - t_3$. Thus, each locus is well-defined by the 3·4 parameters P_1, \dots, P_4 , the 3 timing parameters t_1, t_2, t_3 and one global timing parameter t_{step} , summing up to 16 parameters for one locus. The loci of the front and hind legs are specified separately, but the global time parameter t_{step} is shared, hence the total amount of parameters is 31.

Optimizing these 31 parameters by hand is not feasible and it has become state-of-the-art to use machine learning algorithms to optimize the walk parameters (Kohl & Stone, 2004; Röfer 2004; Hebbel et al., 2006). The criterion to be optimized is the walk speed of the robot. In this case only the straight forward walk is optimized. An individual defines with its 31 object parameters the walk movement and its fitness is evaluated by letting the (simulated)

robot walk for an amount of time and measure its speed. Individuals which result in a walk which is either not straight or lets the robot fall on its side are assigned the worst fitness value, zero. The offsprings have been evaluated in parallel on a compute cluster with 60 nodes, allowing an acceptable simulation time.

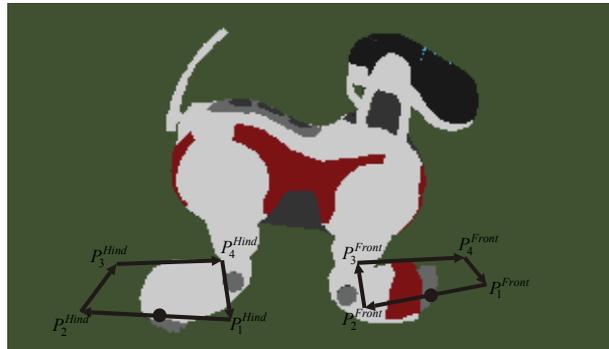


Figure 4. Schematic representation of the leg movement

5. Results

For all experiments the basic settings of the (5,30) Evolution Strategy have been used: the population size of the parents was 5, the size of the offspring generation 30. The selection was a comma selection which ensures a good performance of the self-adaptation of the mutation strengths. The recombination was intermediate and the first parent generation was randomly initialised. Each strategy has been run until 7000 fitness evaluations have been exceeded and repeated with exactly the same settings at least 15 times. The resulting fitness curves of the runs have been averaged to suppress random fluctuations.

At first the performance of the not controlled Model Assisted Evolution Strategies has been compared to the standard Evolution Strategy. The size of λ_p has in this experiment been constantly set to $\lambda_p = 2\lambda = 60$. In case of the polynomial model the 150 closest neighbours to the query point \bar{q} have been used to build the local polynomial model. The Gaussian Process was trained with the 2λ most recently measured fitness evaluations (Ulmer et al., 2003).

Fig. 5 shows the results: all model assisted strategies converge much faster towards an optimum fitness of approximately 66 than the standard Evolution Strategy. All model assisted strategies have found the optimum at the latest after 5000 fitness evaluations. The standard Evolution Strategy however has not reached this optimum after 7000 fitness evaluations. Apparently the capability to model the search space is reflected in the convergence speed. The Gaussian Process model converges fastest closely followed by the polynomial model and the nearest neighbour model.

In the next experiment the effect of the Controlled Model Assisted Evolution Strategy and the λ -Controlled Model Assisted Evolution Strategy was analyzed. For this experiment the polynomial model has been used even though it is slightly inferior to the Gaussian Process model but is a lot faster to be calculated. The adaptation parameters δ_{λ_p} in equation (22) and (23) is set to $\delta_{\lambda_p} = 10$ and δ_{λ} from equation (24) and (25) is set to $\delta_{\lambda} = \mu/2 = 2.5$. The general strategy parameters are set as before. Fig. 6 shows the results of the controlled MAES compared to the not controlled MAES, notice that the ordinate in the graph has a different range than in Fig. 5. Both the C-MAES and the λ -CMAES show a slightly faster convergence

than the uncontrolled MAES. This happens because already after 500 fitness evaluations the model in both cases is good enough to preselect the most promising individuals and thus the impact of the model is increased. A meaningful difference between the C-MAES and the λ -CMAES is not observable for this problem.

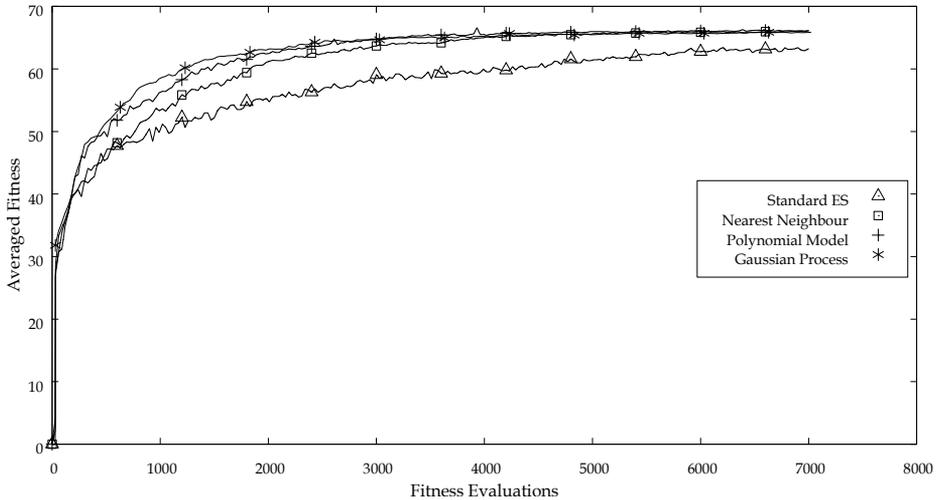


Figure 5. Comparison of the fitness curves of the Standard Evolution Strategy and the Model Assisted Evolution Strategies with different models

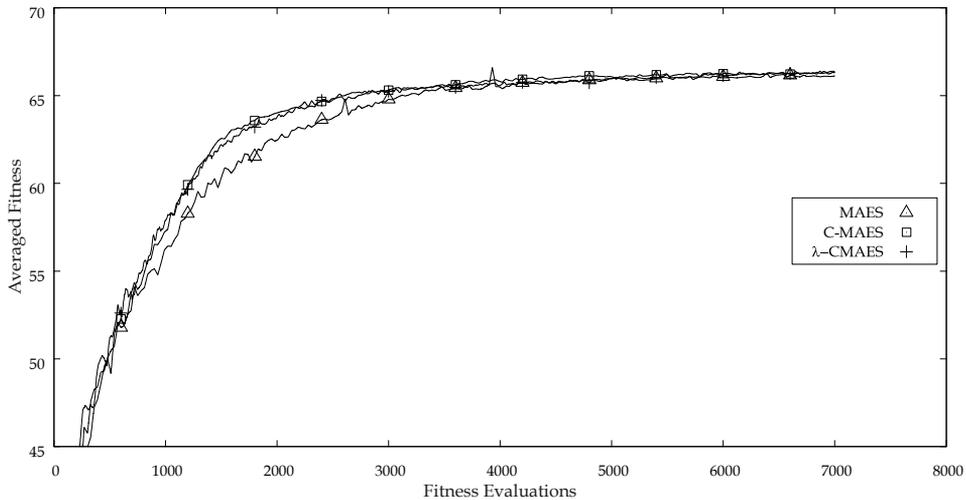


Figure 6. Comparison of the fitness curves of the uncontrolled Model Assisted Evolution Strategy and the controlled Model Assisted Evolution Strategies with different control mechanisms

6. Conclusion

Evolution Strategies are well suited to optimize the parameters of this practical problem. However the MAES outperform the standard ES by far and lead to a much faster convergence. The controlled MAES variants in this case can improve the convergence speed

compared to the uncontrolled strategies a bit more. But since they do not perform worse they should be preferred over the uncontrolled strategies making sure that in case of a bad model quality the model impact is reduced and the Evolution Strategy does not perform worse than a strategy without model assistance.

7. References

- Beyer, H.-G. (1996). Toward a theory of evolution strategies: Self-adaptation, *Evolutionary Computation*, Vol. 3, No. 3, pp. 311-347
- Gibbs, M. & MacKay, D. (1997). *Efficient implementation of gaussian processes*, Technical Report, Cavendish Laboratory, Cambridge, UK
- Hebbel, M.; Kosse, R. & Nisticò, W. (2006). Modeling and learning walking gaits of biped robots, *Proceedings of the Workshop on Humanoid Soccer Robots of the IEEE-RAS International Conference on Humanoid Robots*, pp. 40-48, Genoa, December 2006
- Hebbel, M., Nisticò, W. & Fisseler, D. (2006). Learning in a high dimensional space: fast omnidirectional quadrupedal locomotion. *RoboCup 2006: Robot Soccer World Cup X*, Lecture Notes in Artificial Intelligence, Springer
- Hengst, B.; Ibbotson, D.; Pham, S.B. & Sammut, C. (2002). Omnidirectional locomotion for quadruped robots. *RoboCup 2001: Robot Soccer World Cup V*, pp. 368-373, Lecture Notes in Computer Science, Springer
- Hoffmann, F & Hölemann, S. (2006). Controlled model assisted evolution strategy with adaptive preselection, *Proceedings of 2nd International Symposium on Evolving Fuzzy Systems*, pp. 182-187, Ambleside, September 2006
- Jin, Y.; Hüskens, M. & Sendhoff, B. (2003). Quality measures for approximate models in evolutionary computation, *Proceedings of GECCO Workshops: Workshop on Adaptation, Learning and Approximation in Evolutionary Computation*, pp. 170-174
- Jin, Y. (2003). A comprehensive survey of fitness approximation in evolutionary computation, *Soft Computing*, Vol. 9, No. 1, 2005, pp. 3-12
- Kohl, N. & Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*
- Laue, T.; Spiess, K. & Röfer, T. (2006). SimRobot - A general physical robot simulator and its application in RoboCup. *RoboCup 2005: Robot Soccer World Cup IX*, pp. 173-183, Lecture Notes in Artificial Intelligence, Springer
- Rasmussen, C. E. & Williams, C. K. I. (2006). *Gaussian processes for machine learning*, MIT Press, ISBN 0-262-18253-X
- Rechenberg, I. (1973). *Evolutionsstrategie*. Friedrich Fromm Verlag, Stuttgart
- Röfer, T. (2004). Evolutionary gait-optimization using a fitness function based on proprioception. *RoboCup 2004: Robot Soccer World Cup VIII, LNAL*, Springer
- Schwefel, H.-P. (1975). *Evolutionsstrategie und numerische Optimierung*. Dr.-Ing. Dissertation, Technische Universität Berlin, Fachbereich Verfahrenstechnik
- Ulmer, H.; Streichert F. & Zell, A. (2003). Evolution strategies assisted by gaussian processes with improved pre-selection criterion, *Proceedings of IEEE Congress on Evolutionary Computation (CEC'03)*, pp. 692-699, Canberra, Australia, December 2003
- Ulmer, H.; Streichert, F. & Zell, A. (2004). Evolution Strategies with Controlled Model Assistance, *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pp. 1569-1576, Portland, Oregon, June 2004

Evolutionary Morphology for Polycube Robots

Takahiro Tohge and Hitoshi Iba
University of Tokyo
Japan

1. Introduction

Recent developments in robot technology have led to modular robots that are assembled from multiple parts. Modular robots have pros and cons. Their shape can be changed to suit a particular purpose and the same parts can be reused, which makes them fault-tolerant and extendable. However, such freedom and flexibility lead to higher cost so as to decide the configuration and actions to meet the purpose.

With conventional robots, modular or non-modular, much research has been conducted on the way that a design is determined first and connections with sensory input, internal memory and motor output are acquired through learning. The designs are often based on existing creatures such as insects, dogs and earlier robot models, or parts of such creatures, for example, robot arms. As robot bodies operate in the real world, it makes sense to model them after living creatures that have been successful in nature. In fact, computer simulation patterns akin to living creatures have been created mainly in the field of artificial life, represented by the well-known L-System imitating plants and Dawkins' biomorph (Dawkins, 1986) imitating insects. Sims' virtual creatures (Sims, 1994a; Sims 1994b) feature not only a pattern but also a body and nerves, which are actually stylized art rather than robotics. Modular robots, which are easy to configure and recombine, have an advantage in that optimal robot shapes can be achieved, without modeling the existing creatures. Many existing modular robots are based on the block shape. Although automatic design research is not limited to the block shape, the shape has certain advantages: The basic units are easy to make, handle and combine. Furthermore, blocks are similar to cells, which enabled the simulation of biological development in the research on building structures.

GOLEM Project (Lipson, 2000; Pollack, 2000) is one of the implementation models of Karl Sims experiments. They evolved frame-type virtual creatures using a linear actuator and fabricated them as a robot in a real world. In the modular robotics, new types of robots have been proposed with the functions such as a self-reconfiguration (Murata et al., 2002; Kurokawa et al., 2003) or self-reproduction (Zykov et al., 2005; Griffith et al, 2005). The LEGO Mind Storm is a famous block-based modular robot, parts of which are so small that half-assembled modules are combined in different placements (Lund, 2001). Ideally, robots can actively change their configuration by automatically recombining modules to fit a particular environment. However, this chapter deals with cube-shaped modular robots that can be manually recombined to achieve significant configurations that can adjust to actual environments using evolutionary morphology. This chapter describes how successfully EC

is applied to the morphology for real cubic robots. More precisely, we empirically show the following points by several experiments:

- EC is effectively applied to designing the morphogenesis of cubic robots.
- Evolutionarily designed morphogenesis proves to be superior to manually designed robots in terms of the performance, e.g., speed or stability.
- The effectiveness of our approach is demonstrated by testing the generated behavior with a real robot, i.e., ROBOCUBE.

The rest of this chapter is organized as follows. The next section introduces the ROBOCUBE we have used as a kind of cubic modular robot. Section 3 describes two proposed methods of the evolutionary morphologies. Section 4 and 5 present the experimental results to show the effectiveness of our approach. The discussion of these experiments is provided in Section 6. Finally, Section 7 concludes the chapter.

2. ROBOCUBE

A block-based modular robot, ROBOCUBE, is used for the experiment. A ROBOCUBE block is a 5cm³ cube, each face of which has four connectors to be connected by button-shaped contact points to interlock the power supply line and the communications line to automatically establish a network for each module. Blocks are classified as sensor-related blocks (e.g., supersonic waves, light, sense of touch, gyro), motor blocks, output-related blocks (e.g., emission of light, buzzer, relay) and wireless blocks, which can provide wireless connectivity. Although one core block and one power source are essential for one robot body, once they are available, the robot is activated regardless of the connection; therefore, an enormous number of combination patterns exist. The number of blocks required to activate the actual robot is limited, and in case a block with certain functions is used, the maximum number of blocks to be used for the whole body is 16. The actuator for ROBOCUBE is a motor block with a DC servomotor embedded in it. The motor block has a turning-angle sensor for velocity and position controls and is used for movement with wheels or to make a turn by connecting to a sensor block. However, connecting such functional blocks necessitates wireless blocks or cables because each line is split at the motor axes.



Figure 1. A sample of ROBOCUBE

3. Morphogenesis

We used the following models for robot morphogenesis. The first model used Messy GA (Goldberg, 1989) and cellular automaton, whereas the second adopted GP (ANT). The first model easily configures a lump mass with a method modeling the development of living creatures, and the second effortlessly achieves a string-shaped structure with a method using the ANT route.

There are two restrictions on morphogenesis for ROBOCUBE: the body must neither split nor interfere. The restrictions interact, and eliminating a block in a certain position to avoid interference creates the possibility of decoupling. There are several methods of expressing morphology via a certain kind of coding, however, the important issues are the fitness of the evolutionary computation and how to easily avoid the above restrictions. Each model is detailed below:

3.1 MODEL Using Messy GA

This development model is based on the conditions for connecting with adjacent blocks. The development model, or the cellular automaton that never dies, is a set of conditions to determine in which positions to connect a new block according to how the surrounding blocks are connected.

Think of a simple 2-D case. Focusing on one block, there are four adjacent dimensions: top, bottom, left and right. Four flags, which correspond to each direction, are prepared both for the condition and the action parts. If the flag for the condition part is 1 (meaning true), the block must be connected in that direction. A 0 (meaning false) flag shows that the block must not be connected in that direction. When the condition part fits in each of the four directions, a block is connected in a direction for which the condition part flag is 1.

This is the basic idea of the development model based on the conditions for connection. However, in reality, various extended models exist by adding other conditions including a coding method. Use of the development concept helps to reduce the parameters for determining the robot design and works well with the evolutionary method. This model can also easily resolve the interference issue by introducing dummy blocks. To insert a motor block, for example, dummy blocks are aligned along a trajectory of moving parts to prevent other blocks from being inserted in positions that have the possibility to interfere. (Nakano et al, 1997; Asai, 2002) is part of the research on this development method for actual robots.

The easiest method for implementing this development model is to prepare a set of rules covering all these conditions. One development condition is obtained given a character string $16 \times 4 = 64$ bit if the position of the character string is considered to be the condition part. The 0th 4 bit is an action part corresponding to 0000 of the condition part, the first 4 bit is an action part corresponding to 0001 of the condition part and so on. The merit of this method is that only character strings of a fixed length are necessary (64 bit), which allows easy adjustment to a genetic algorithm. A risk of this method is that a small number of genes could determine the final morphology. The genes of the former half, for which the condition parts are 0, are highly influential, therefore the location of each gene is a key factor and a 64 bit chromosome has only 16 genes.

To solve this problem, we consider providing more than one pair of a set of conditions and action parts, i.e., a basic model and its corresponding rule. For instance, the length of a chromosome of an 8 bit gene consisting of a 4 bit condition part and a 4 bit action part is 8

bit * N. This rule corresponds to a gene of Messy GA, checks the condition part from the front gene and starts connecting blocks based on the action part of the first gene that meets the condition. Each gene is moved from front to back and from back to front by crossover; therefore, a dependency on the location of genes is reduced for the aforementioned model.

This model may have the disadvantage of developing slowly because the small size of the genes can cause difficulty in finding a corresponding condition part. In this case, introducing “*:don’t care” (meaning that the conditions for a connection are disregarded for the location) would solve the issue of a smaller number of corresponding condition parts. However, the action parts for genes of variable length are still 4 bit and therefore the condition parts containing many 1s make the corresponding action part meaningless. For example, an insignificant rule such as the action part 1100 corresponds to the condition part 1100 could be generated. Although such an insignificant action part can be considered an intron in a chromosome, the action part is disassembled as follows by increasing the condition parts to reduce the impact of one gene on the entire development process.

Genes originally have a one-directional condition part. Now focusing on one block in the development stage, to connect a new block with the block, an opening must exist in at least one of the four directions. The directional condition part leaves it to the manifestation condition of the genes. When conditions for connection other than the directions written on the rearward condition part of a gene are confirmed and met, the manifestation of the gene occurs and it adds additional blocks (i.e., action) in the direction of the directional condition part. In case of two dimensions, for instance, if the bottom is selected as the directional condition part, additional blocks are connected in the eight patterns as shown in Fig.2. This method reduces dependency on one gene and therefore the size of a chromosome, i.e., the number of genes must be increased to some extent. In that case, dummy blocks, which are assumed for the interpretation of directional conditions, are not considered to exist when conditions for connection are interpreted.

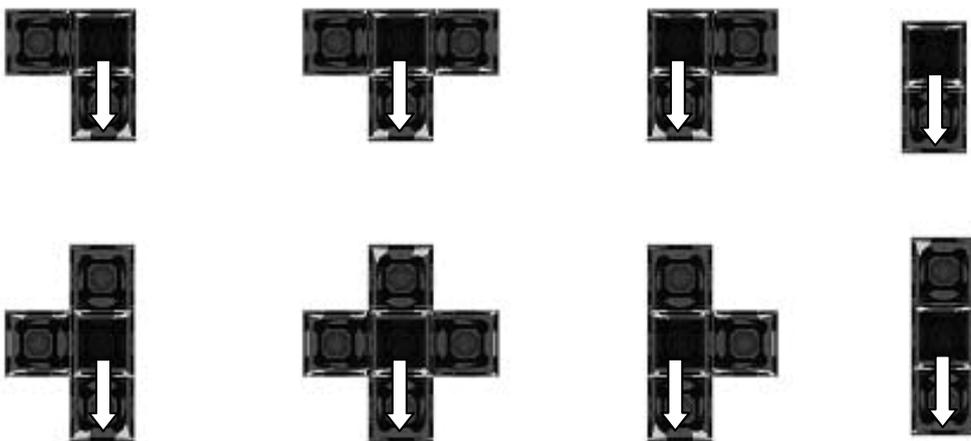


Figure 2. The eight development patterns in 2-D (Each white arrow means a new connection)

As only simple conditions for connecting cellular automaton are expected to lead to a monotonous development process, including the manifestation conditions by the age of the block during the development process in what step the block was connected can ramify the blocks. Considering the above, the G-TYPE is defined as a gene of Messy GA, which is 3-D as in the experiment.

Condition			Action
Direction	Connection	Step	Kind

- Direction: directional conditions (top, bottom, left, right, forward, backward)
- Connection: conditions for connecting in the five rest directions (5 bit)
- Step: manifestation occurs when the age of the block is less than the step (from 0 to the maximum size of the block)
- Kind: kinds of blocks (normal, motor, sensor)

Considering the condition parts as the locus, the use of such genes can cause Messy GA evolution. The actual procedure is as follows:

1. Initialize chromosomes at random to generate the initial population.
2. Morphogenesis for the initial-state robot by gene manifestation.
3. Morphogenesis ends when the genes for manifestation run out or the specified number of blocks is reached.
4. Implement a task to determine the fitness.
5. Generate the next-generation population through selection, crossover and mutation.
6. Return to No. 2 above.

3.2 Models Using GP

In these models, the Agent (ANT) aligns the functional blocks of module-combination-type robots in 3-D. That is, an agent exists that can take actions such as “turn right,” “turn left,” “look up,” “look down” and “put a block and go forward,” and the morphology that is formed when the agent has finished aligning becomes that of a robot that implements the task. As an agent must align the blocks before it moves, the blocks are continuously aligned. In 3-D, the length is expressed by the x axis, the width by the z axis and the height by the y axis. Only two kinds of blocks are used: the motor block, which can rotate in any arbitrary angle, and the normal block, which cannot. Agents directly align only normal blocks, among which those that are already aligned and meet the following conditions are converted into motor blocks.

$$d_{xz}(RG;BG) > 1 \quad (1)$$

$$l = \max(d_{xz}(RG;BG)) \cdot r \quad (0.1 < r < 0.9) \quad (2)$$

The equation $d_{xz}(a; b);RG;BG$ expresses the Euclidian distance, the gravity point of a robot and that of a block, respectively, with regard to a and b on the xz plane. The terminal and non-terminal nodes of GP used in this paper are shown in Tables 1 and 2. The expression “allows connection of motor” refers to whether connecting with a block that serves as a supporting point of motor rotation is possible. If there is more than one candidate that can be connected, the plus (+) directions of the z, y and x axes are searched sequentially and connected to the first block that is found. A motor block is vertically connected to the z axis. After determining where the motor is to be connected, the normal blocks that interfere with the rotation are eliminated. A body is formed in this way, and the GP procedure is applied to the balance.

Function name	Definition
Prog2	Take two terminal nodes and implement them in order.
Prog3	Take three terminal nodes and implement them in order.

Table 1. Non-terminal nodes

Function name	Definition
Right	Turn right
Left	Turn left
Up	Turn up
Down	Turn down
Put and move 1	Put a normal block that allows connection of motor and move forward.
Put and move 2	Put a normal block that does not allow connection of motor and move forward.

Table 2. Terminal nodes

4. Experiment 1: Moving the plane surface

4.1 Setting of the Environment

The first experiment attempts the evolutionary morphology of the working leg parts instead of wheels, without using sensors. A plane surface is assumed, and a robot proceeds where there are no obstacles. The initial state is a straight line of core blocks between two motor blocks that has one block as the leg part as shown in Fig.3. Even though the robot moves forward in this state, its rotating block touches the floor for such a short time that it moves only incrementally.

The Messy GA + CA method was applied to morphogenesis. A maximum of 10 blocks were used for the leg part. The core block has two rollers to help reduce the impact of friction on the simulator. However, the rollers can become meaningless objects or even obstacles depending on the configuration of the robot. Controllers to output constant rotations and sin waves should be prepared in advance to define the following action genes in a chromosome.

The action genes are initiated at random and exist in chromosomes at a constant probability. If an action gene corresponds with a certain motor, the action is output; otherwise, no action occurs.

Fitness is computed by the moving distance on the plane [cm] multiplied by the stable movement time, and the simulation is conducted up to a maximum of 5[s]. If a controller does not exist, fitness is computed by reducing the maximum number of blocks (10) from the number of blocks of the leg part. Better fitness is obtained where more blocks are used.

The following GA parameters were used. The Messy GA method used cut and splice for the crossover. If the length of a chromosome exceeds 1000 due to crossover, the experiment is repeated from the selection.

- Number of individuals: 128
- Number of generations: 512
- Length of the initial chromosome: 20
- Maximum length of a chromosome: 1000
- Probability of mutation: 0.05

- Method of selection: tournament method (size 4)
- Generation Gap: 0.1

The experiment was made using a dynamic 3-D simulator-loading engine as in an actual environment, and the actual machines were used to ensure functionality of the obtained individuals.



Figure 3. The initial structure and the movement (Plane surface)

4.2 Result

Fig.4 shows the maximum and average fitness for each generation at the time of action. The "wheel" means the fitness of a normal wheel robot (Fig.7), of which we can think as a target example for this moving task. Fig.5 shows some of the individuals that emerged during the evolution, and Fig.6 shows the second optimal individual resulting from the evolution. The optimal individual uses the leg part as a support during rotation, and the central rollers handle the rotating movement. As you can see from 4, the fitness of the best individual (e.g., Fig.6) is superior to the normal wheel robot. We performed the same experiment with the actual machines and observed the behavior in the real world.

It may be easy to imagine individuals that adopt a controller of constant rotation, maximizing the diameter of the leg parts, however, an individual as in Fig.6 was actually obtained. The moving distance was maximized by rotation using the rollers attached to the body, which was realized by spreading the leg part in one direction. In reality, the individuals that maximize the diameter, such as the one with the cross-shaped feet, cannot take advantage of friction as a driving force, thereby spinning around and going nowhere.

Table.III shows the fitness values of evolved individuals. Fitness is measured as the distance from the initial position at the end of simulation. Note that the performance of the suboptimal robot is 150% better than the wheel robot. The cross-shaped robot (see Fig.7) was stuck by its edge and could barely move. As a result of this, its fitness value is very small. This table clearly shows the effectiveness of the proposed evolutionary morphology in terms of the fitness values.

During evolution, some were seen to move forward, leaping lightly by generating asymmetrical leg parts to take sin-wave output. From the viewpoint of the practical movement method of modular robots, they can be classified into those that transform like ameba to change a whole location and those that use leg parts or wheels. Although this result is an option due to a higher velocity than wheels, it requires sufficient strength and a function to shift directions.

In this model, each gene does not have significant impact and therefore the initial state must contain various genes, which is made possible by increasing the number of individuals and

the initial chromosome length. In the experiment, the fitness computation from implementing the tasks was so costly that the initial chromosome length had to be extended. As this issue tends to increase the selection pressure, we tried to avoid it by reducing the generation gap. However, the average fitness was actually apt to shrink as if it was pulled by the optimal individual.

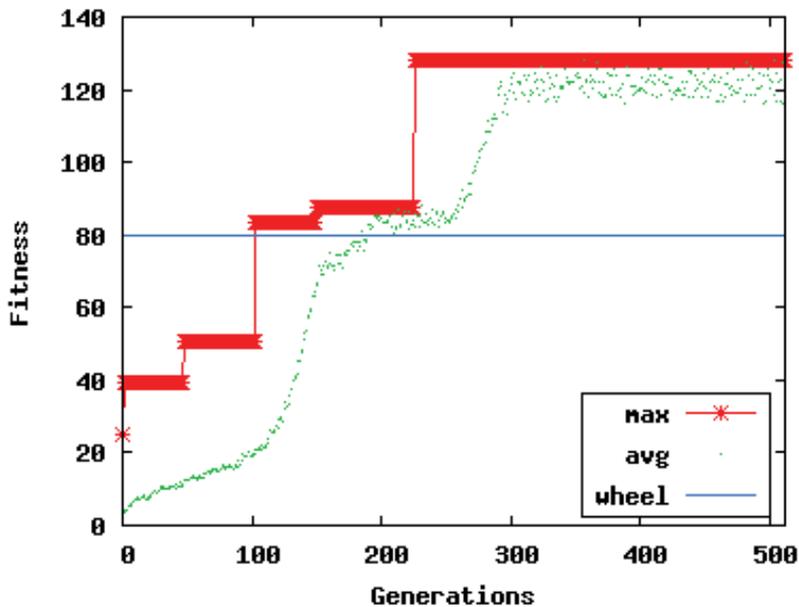


Figure 4. Change in fitness (Plane surface)

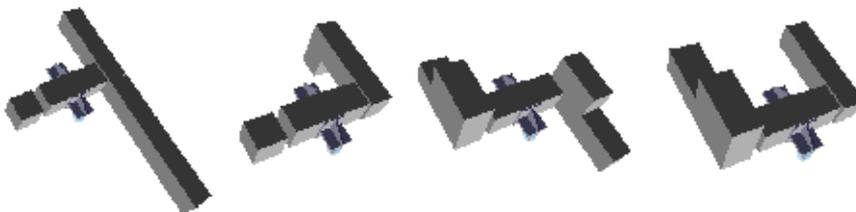


Figure 5. The suboptimal individuals in each generation (Plane surface)



Figure 6. The best individual and the movement (Plane surface)

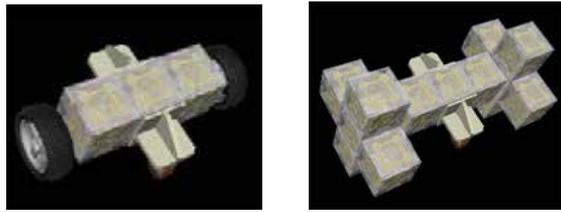


Figure 7. The wheel robot and the cross-shaped robot

Type	Fitness
Best	127.94
Wheel	80.83
Initial	4.68
Cross-shaped	2.46

Table 3. Fitness values

5. Experiment 2: Ascending a slope

5.1 Setting of the Environment

This experiment attempted to ascend stairs via morphogenesis using GP. Fig.8 shows the stairs used in the experiment. The stair height was set higher than the usual wheel forward movement. The colored part shows the stairs that the robots ascended. The stair width was 2 m, exceeding which meant the robot falling and terminating the attempt. Staircases were located 1 [m] away from the initial location of the robot in each of the four directions. The direction of the robot's initial location did not matter.

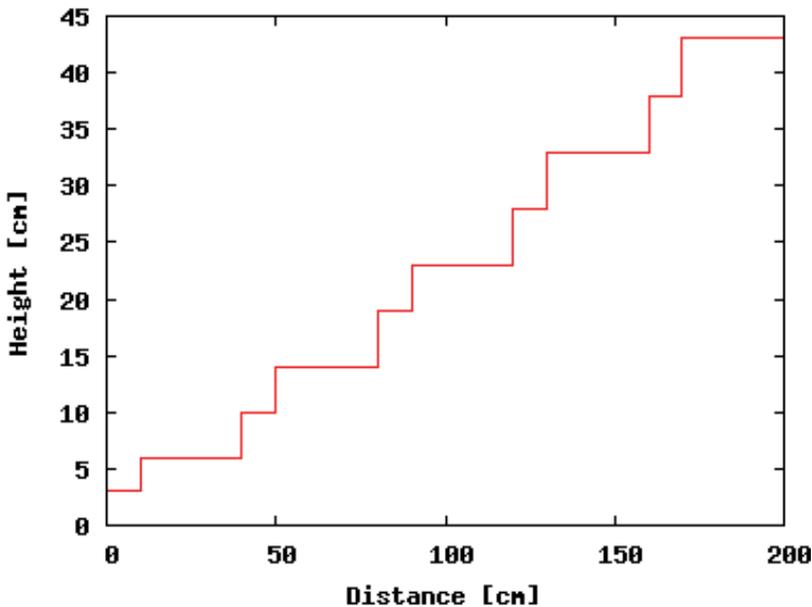


Figure 8. Section of a staircase

The controllers were not changed, and the robots were made to rotate under a certain rule in a constant direction. The initial evaluated value is 1500, and the value is reduced by 500 when the robot reaches the first stair and by 100 for each subsequent stair that the robot ascends. The value of the trial time divided by 10 is added at the end of the trial. The maximum trial time was 1000. The less the evaluated value is, the better. If no normal blocks emerge during the action, an evaluated value of 3000 is given.

The following parameters were used:

- Number of individuals: 250
- Number of generations: 50
- $r = 0.9$
- Selection method: Tournament method (size 6)

5.2 Result

Fig.9 shows the minimum fitness and the average fitness for each generation at the time of implementing GP. Fig.10 shows the good individuals that emerged in each generation. The light-colored blocks in the center are normal blocks, and the dark-colored blocks at the edge are motor blocks.

Fig.10(a) shows an individual that emerged in the third generation. Because only one motor emerged, which made proceeding straight ahead difficult, it was not able to achieve a task. Fig.10(b) shows an individual that emerged in the eighth generation, featuring five blocks vertically aligned.

The robot supported itself using its long vertical part and got on the gap between the stairs, but the tilt could not be supported. Afterward, the robot deviated from the right orbit. The individual that emerged in the 14th generation, shown in Fig.10(c), ascended the stairs, supported by the five central blocks as well. Starting with the eighth generation, there were two more blocks at the front. The wider shape could avoid deviation from the orbit caused by the tilting at the time of climbing over the gap in the stairs.

Fig.11 shows the best individual during the simulation, which emerged in the 16th generation. Putting the three blocks on the right-side front of the robot upon the next stair, the robot could start climbing the staircase. Compared with Fig.10(c), there were fewer front blocks, thereby eliminating the front block obstacles and allowing the robot to ascend over the gap in the stairs and shorten the time to realize the task.

6. Discussion

We showed two experiments with different coding methods, i.e., messy GA and GP. In modular robotics, it is very important to acquire not only one best pattern but also various sub-optimal patterns. This is because modular robots are used and developed for the sake of search or rescue task, which requires the flexibility against the real-world environment. In such a case, moving is a basic and essential ability skill.

Our experimental results have shown how successfully the significant pattern was acquired during the evolution. As can be seen from Fig.4 and Table.3, some of those patterns were superior to those designed manually for the target task.

In this experiment, sensors were not used to avoid complicating controllers. Introducing sensors necessitates a certain level of dynamic generation of controllers and symbiosis with morphology models. Some computation cost is necessary, for which an application of

Neural Net or Q-Learning is a solution. Research in this field includes the co-evolution of morphology and behavior as well as the automatic generation of controllers. However, we intend to deepen our knowledge of expression methods of morphology for actual robots, and in that regard we will attempt the co-evolution of vegetable-type robots using only sensors and animal-type robots with motors.

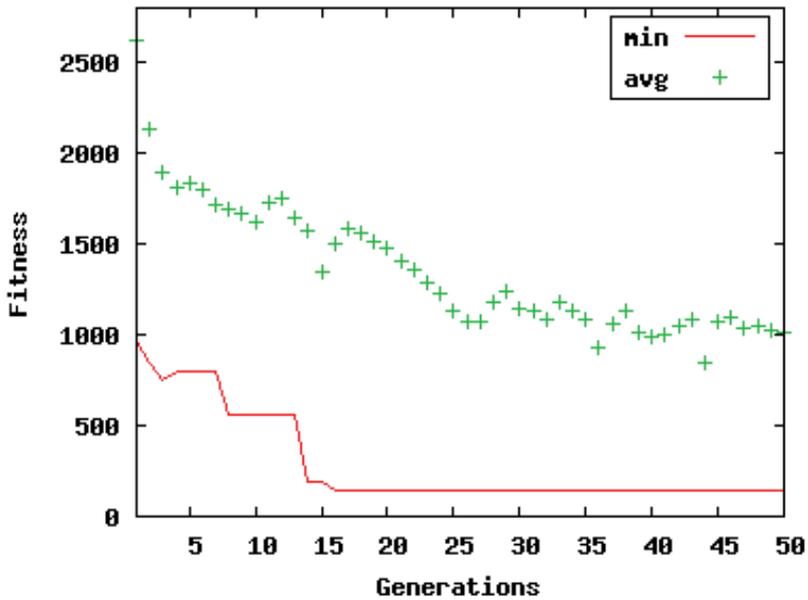


Figure 9. Change in fitness (Slope, less is better)

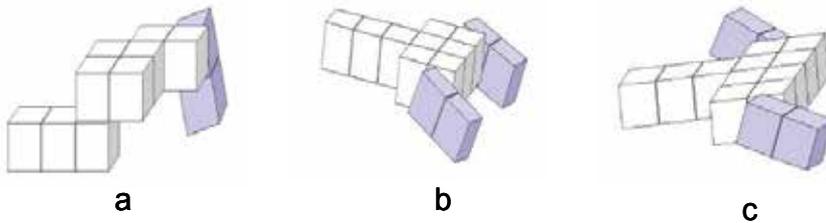


Figure 10. The suboptimal individuals in each generation (Slope)



Figure 11. The best individual (Slope)

7. Conclusion

The chapter covered the evolutionary morphology of actual robots using a 3-D simulator. An individual fitted for plane movement using Messy GA rotated its body by extending the leg parts in a direction perpendicular to the rotation axis to increase stability. The individual that fitted itself to the ascending stairs by GP adjusted well to the stair gap by extending the support on the back of the body in a flat and a hook shape. Either shape is typical of block-type robots, but uncommon for humans, and reflects the versatility of blocks as a basic unit. A future issue will be to apply to more complicated tasks or multi-agent cooperation using modular robotics.

8. Acknowledgements

The authors thank Kenta Shimada and Taiki Honma for helpful comments and for other contributions to this work.

9. References

- Asai, Y. & Arita, T. (2002). Coevolution of Bodies and Behavior in a Modular Reconfiguration Robot, *IPSJ Journal (in Japanese)*, Vol. 43, No. SIG10, pp. 110-118
- Dawkins, R. (1986). *Blind Watchmaker*, Longman
- Goldberg, D. E.; Korb, B. & Deb, K. (1989). Messy Genetic Algorithms: Motivation, Analysis, and First Results, *Complex Systems*, vol. 3, pp.493-530
- Griffith, S.; Goldwater, D. & Jacobson, J. M. (2005). Robotics: Self-replication from random parts, *Nature*, Vol. 437, pp. 636
- Kurokawa, H. et al. (2003). M-TRAN II: Metamorphosis from a Four-Legged Walker to a Caterpillar, *Proceedings of International Conference on Intelligent Robots and Systems (IROS 2003)*, pp.2452-2459
- Lipson, H. & Pollack, J. B. (2000). Automatic Design and Manufacture of Artificial Lifeforms, *Nature*, Vol. 406, pp.974-978
- Lund, H. H. (2001). Co-evolving Control and Morphology with LEGO Robots, *Proceedings of Workshop on Morpho-functional Machines*
- Murata, S. et al. (2002). M-TRAN: Self-Reconfigurable Modular Robotic System, *IEEE/ASME Trans. Mech.* Vol. 7, No. 4, pp. 431-441
- Nakano, K. et al. (1997). A Self-Organizing System with Cell-Specialization, *IEEE International Conference on Evolutionary Computation*, pp.279-284
- Pollack, J. B. & Lipson, H. (2000). The GOLEM Project: Evolving Hardware Bodies and Brains, *The Second NASA/DoD Workshop on Evolvable Hardware*
- Sims, K. (1994a). Evolving Virtual Creatures, *Proceedings of Computer Graphics*, pp.15-22
- Sims, K. (1994b). Evolving 3d morphology and behavior by competition, In R. Brooks and P. Maes, editors, *Proceedings of the International Conference Artificial Life IV*
- Takahiro, T; Shimada, K. & Iba, H. (2006). Evolutionary Morphology for Cubic Modular Robot, *Proceedings of 2006 IEEE World Congress on Computational Intelligence*
- Zykov, V. et al. (2005). Self-reproducing machines, *Nature*, Vol. 435, pp.163-164

Mechanism of Emergent Symmetry Properties on Evolutionary Robotic System

Naohide Yasuda, Takuma Kawakami, Hiroaki Iwano,
Katsuya Kanai, Koki Kikuchi and Xueshan Gao
Chiba Institute of Technology
Japan

1. Introduction

In order to create an autonomous robot with the ability to dynamically adapt to a changing environment, many researchers have studied robotic intelligence, especially control systems, based on biological systems such as neural networks (NNs), reinforcement learning (RL), and genetic algorithms (GA) (Harvey *et al.*, 1993, Richard, 1989, and Holland, 1975). In a recent decade, however, it has been recognized that it is important to design not only robotic intelligence but also a structure that depends on the environment as it changes because the dynamics of the structural system exerts a strong influence on the control system (Pfeifer & Scheier, 1999, and Hara & Pfeifer, 2003). The behavior of a robot is strongly affected by the physical interactions between its somatic structure and the outside world, such as collisions or frictions. Additionally, since the control system, the main part of robotic intelligence, is described as a mapping from sensor inputs to actuator outputs, the physical location of the sensors and actuators and the manner of their interaction are also critical factors for the entire robotic system. Therefore, to design a reasonable robot, it is necessary to consider the relationship between the structural system and the control system, as exemplified by the evolution of living creatures.

From this point of view, several researchers have tried to dynamically design structural systems together with control systems. Sims (Sims, 1994) and Ventrella (Ventrella, 1994) have demonstrated the evolution of a robot with a reconfigurable multibody structure and control system through computer simulation. The Golem Project of Lipson and Pollack has realized the automatic design and manufacture of robotic life forms using rapid prototyping technology (Lipson & Pollack, 2000). Salomon and Lichtensteiger have simulated the evolution of an artificial compound eye as a control system by using NNs and have shown that the robot creates motion parallax to estimate the critical distance to obstacles by modifying the angular positions of the individual light sensors within the compound eye (Salomon & Lichtensteiger, 2000). These researches have shown the importance of adaptation through not only intelligence but also the relationship between morphology and intelligence. However, the mechanism of the function emerging from such relationship or some kind of design principle is not fully understood yet.

Meanwhile, for living creatures, symmetry properties may be a common design principle; these properties may have two phases, that is, the structural and functional phases. For

example, most legged creatures are symmetric in the structural phase and their gait, that is, the manner in which they actuate their left and right legs, is also symmetric in the functional phase. For the locomotion of a biped robot, Bongard *et al.* have demonstrated the importance of a symmetry structure from the viewpoint of energy efficiency (Bongard & Paul, 2000, and Bongard & Pfeifer, 2002). This is an example of effective symmetry structure from the viewpoint of engineering. However, the effectiveness of an asymmetry structure has also been shown in nature. Although insect wings to fly are symmetric, those to sing are generally asymmetric. One claw of the fiddler crab is extremely big as compared with another. The asymmetric brain structure of a fruit fly enhances its long-term memory (Pascual *et al.*, 2004) and an asymmetric ear structure of barn owls allows accurate auditory localization (Kundsén, 2002). These examples indicate that since living beings must have created optimal mechanisms through interactions with the environment, the characteristics of symmetry or asymmetry are extremely important for not only the physical structure but also functionality, including control. Hence, since the symmetry properties and their concomitant functionality show the design principle of the entire system, the clarification of the mechanism of the emergence of symmetry properties can contribute to the development of a methodology for a robotic system that designs its own morphology and intelligence depending on the changing environment.

From this point of view, we have studied the mechanism of symmetry properties emerging from the balance between structural and control systems by using an evolutionary robotic system with reconfigurable morphology and intelligence (Kikuchi & Hara, 1998, Kikuchi *et al.*, 2001, and Kikuch & Hara, 2000). Here, as an example of our studies, we introduce the symmetry properties created by two relative velocity conditions, fast predator vs. slow prey and slow predator vs. fast prey, and by genotype-phenotype noise conditions, genetic errors due to a growth process.

2. Task and Evolutionary Robotic System

In this section, we introduce a task for a robot, a fitness criterion, and an evolutionary robotic system.

2.1 Task and Evaluation

The task given to the robot is to maintain a certain distance D from a target. The robot and the target are in an arena surrounded by walls, as shown Fig. 1. The target moves randomly and the robot behaves by using the morphology and intelligence automatically generated by genetic programming (GP). Note that, the short distance D means that since the robot chases the target, the predator chases the prey. On the other hand, the long distance D means that since the robot departs from the target, the prey runs away from the predator.

A fitness value, F , is calculated according to the performance of the robot. The performance is evaluated by using a multiobjective function that is defined as

$$F = \frac{1}{N} \sum_{i=1}^N \left\{ \frac{1}{T} \int_0^T \alpha (\|P - X\| - D)^2 dt \right\}$$

$$\begin{cases} (\|P - X\| - D) \geq 0 & \alpha = 1 \\ (\|P - X\| - D) < 0 & \alpha = \frac{\sqrt{2}H}{D} \end{cases} \quad (1)$$

where X is the center of the robot, P is the center of the target, t is the time, T is the total evaluation time, H is the side length of the arena, i is the trial number, and N is the total number of trials. The robot obtains a high evaluation if it maintains D . Here, the weight α is determined by the distance between the robot and the target. When this distance is smaller than D , α is $\sqrt{2}H/D$, and when it is larger than D , α is 1. Note that the value of $\sqrt{2}H$ means the maximum distance of the robot and the target. Additionally, the smaller the fitness value, the better the performance. When the robot collides with the target, the fitness value is 1.63 and when the robot maintains an objective distance, it is 0.0.

2.2 Evolutionary Robotic System

The robot is modeled as a cylindrical shape and has two visual sensors and two wheels with motors. The motion is calculated on the basis of a two-dimensional dynamic model that includes realistic physical conditions such as collisions and frictions. The equations of motion are given by

$$\begin{aligned} M\ddot{x} &= \sum_{i=0}^1 \frac{T_i}{R_i} \cos \theta + F_x + P_x \\ M\ddot{y} &= \sum_{i=0}^1 \frac{T_i}{R_i} \sin \theta + F_y + P_y \\ I\ddot{\theta} &= \sum_{i=0}^1 \frac{T_i}{R_i} r + F_\theta + P_\theta \end{aligned} \quad (2)$$

where M is the mass of the robot, x and y are the coordinates of the center of the robot, T_i is the torque of the motor, R_i is the wheel radius, r is the distance from the center of the wheel to the center of the robot (equals to the robot radius), F_* is the friction with the floor, P_* is the impact with the target or a wall, I is the moment of inertia, θ is the direction of the robot, and i is the wheel ID that is 0 for the left wheel and 1 for the right wheel, as shown in Fig. 2. Note that the origin is the center of the arena and the counterclockwise direction is positive, as illustrated in Fig. 1. Using these equations, the motions of the robot and target are simulated by a Runge-Kutta method.

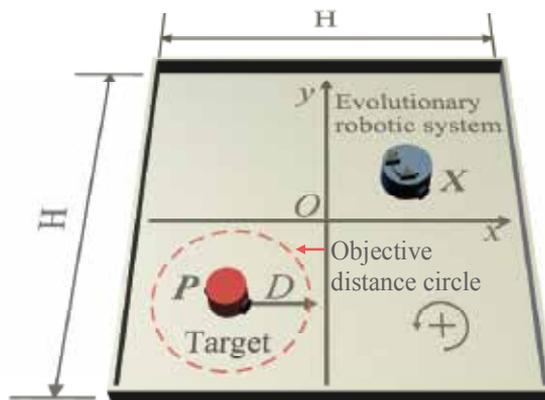


Figure 1. Simulation arena

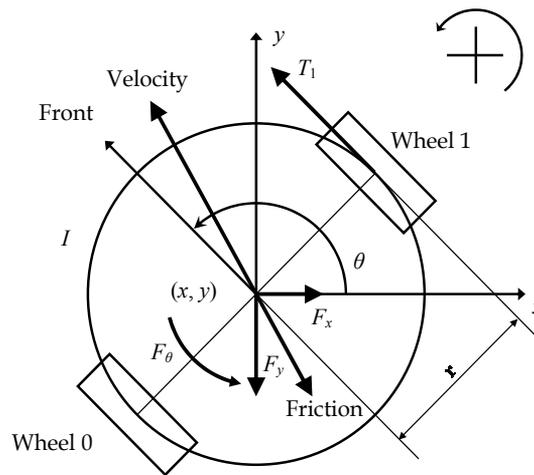


Figure 2. Two-dimensional model of evolutionary robotic system

3. Morphology and Intelligence Genes

In this study, the evolutionary robotic system is optimized through the processes of GP: (1) development, (2) evaluation, (3) selection and multiplication, and (4) crossover and mutation. Under GP, each robot is treated as an individual coded by a morphology gene and an intelligence gene. In this section, we explain the coding method.

3.1 Morphology Gene

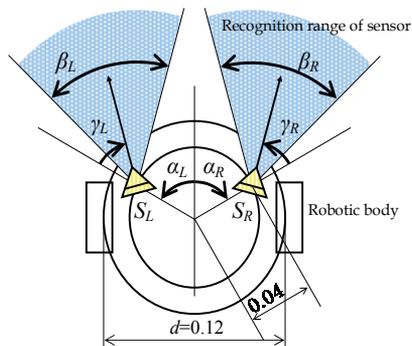


Figure 3. Morphological parameters

A morphology of a robot may be generally defined by using many kinds of elements such as the body shape, size, weight, rigidity, surface characteristics, and sensor-actuator arrangement. In this study, the morphology is represented by the physical arrangement of two flexible visual sensors, two fixed motors, and a cylindrical shape, as illustrated in Fig. 3. Here, two visual sensors S_L and S_R have three degrees of freedom: alpha, beta, and gamma. Alpha corresponds to the arrangement angle of the sensors on a circumference of a circle with a radius of 0.04 m ($0^\circ \leq \alpha_L, \alpha_R \leq 90^\circ$), beta is the range of the field of view

($0^\circ \leq \beta_L, \beta_R \leq 50^\circ$), and gamma is the direction of the visual axis ($-90^\circ \leq \gamma_L, \gamma_R \leq 90^\circ$). Thus, the evolutionary robotic system has six degrees of freedom for the morphology gene. Note that the shaded areas show the recognition areas for the target; the sensor becomes "ON" when the target is recognized in this area. The sensor resolution is set to be 1 for simplicity.

3.2 Intelligence Gene

The intelligence gene of the robot is a computer program described as a decision tree that represents the relationship between the sensor inputs and the motor outputs. The decision tree is created by using two kinds of nodes--terminal nodes and nonterminal nodes--as shown in Table 1. The terminal nodes are the sensor nodes and motor nodes. The sensor nodes *L* and *R* correspond to the state of the two sensors S_L and S_R shown in Fig. 3, with "true" and "false" assigned to "ON" and "OFF." The motor nodes have the action functions such as *MOVE_F* to move forward, *TURN_L* to turn left, *TURN_R* to turn right, *MOVE_B* to move backward, and *STOP* to stop. Figure 4 shows the behavior of these functions. The nonterminal nodes are function nodes, i.e., typical computer language commands such as *IF*, *AND*, *OR*, and *NOT*. The robotic intelligence gene is automatically created by combining these nodes.

Terminal nodes	
Sensor nodes	<i>L, R</i>
Motor nodes	<i>MOVE_F, TURN_L, TURN_R, MOVE_B, STOP</i>
Nonterminal nodes	
Function nodes	<i>IF, AND, OR, NOT</i>

Table 1. Node for decision tree

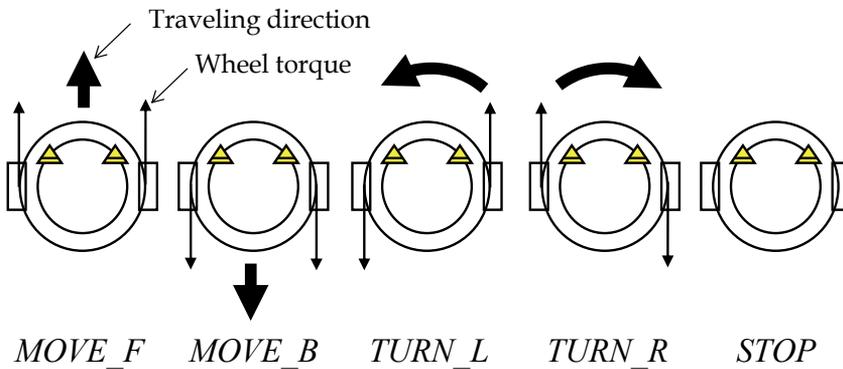


Figure 4. Robotic behaviors for each motor node

4. Evolutionary Simulation I

4.1 Conditions for Simulation

In this study, to clarify the mechanism of emergent symmetry properties, we performed two simulations for different relative velocities of the robot: in Case A, the robot was twice as fast as the target and in Case B, the target was twice as fast as the robot. Since we set an

objective distance D as a short distance of 0.5 m, the robots mean the fast predator in Case A and the slow predator in Case B.

The physical conditions were as follows. The length of one side of the arena H was 4.0m, the diameter of the robot and target d was 0.12 m, the evaluation time T ranged from 20.0 s to 90.0 s, the maximum speeds of the robot and target were 0.2 m/s and 0.1m/s, respectively, in Case A and 0.1 m/s and 0.2 m/s, respectively, in Case B, the sampling time of the sensor was 100 ms, and the weight of the robot and the target M was 0.4 kg. The recognition error of the sensors was set from -3.0° to 3.0° (randomly determined from a normal distribution). The GP parameters were set as follows. The population size was 300, the generation was 300, the selection ratio was 0.8, the crossover ratio was 0.3, and the mutation ratio was 0.1. The initial positions and directions of the robot and target were randomly determined from a uniform distribution within the center region.

4.2 Definition: Indices of Symmetry Properties

To analyze the structural symmetry properties of the robotic system, we defined three indices: $|\alpha_L - \alpha_R|$, $|\beta_L - \beta_R|$, and $|\gamma_L - \gamma_R|$. Hence, the smaller the indices, the higher was the structural symmetry. In the development of the first GP process, these values were uniformly generated to avoid bias.

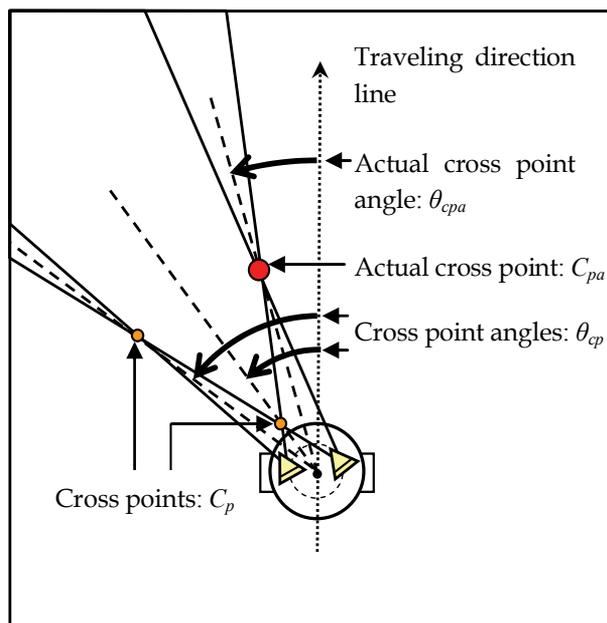


Figure 5. Definition of cross-points and cross-point angles

Additionally, we defined another index for the state space created by the visual sensors. As illustrated in Fig. 5, the values C_p represent the cross-points of the recognition areas of the two visual sensors, and the values θ_{cp} represent the angle between the traveling direction line and the line connecting the cross-points and the center of the robot. Note that the maximum cross-point number is four, since each visual sensor has two edges of recognition

area. We further defined the cross-point that is employed for action assignment as an actual cross-point C_{pa} . Similarly, θ_{cpa} represents the actual cross-point angle.

Using these parameters, we performed 20 simulations for each case and analyzed elite individuals in the final generation of each simulation.

4.3 Results

Table 2 shows the fitness averages of the elite individuals obtained in Cases A and B and the standard deviations. The fitness in Case A is better than that in Case B, since the robot is faster than the target and can quickly approach it. Here, the fitness value of 0.218 means that the robot departs averagely 0.14 m inside from the objective distance circle shown in Fig. 1, and 0.278 means that it departs averagely 0.16 m inside.

	Case A	Case B
Ave.	0.218	0.278
Std. dev.	0.056	0.086

Table 2. Fitness in Cases A and B

Morphology genes	$\alpha_R=65, \beta_R=35, \gamma_R=-32$
[deg]	$\alpha_L=37, \beta_L=27, \gamma_L=64$
Intelligence genes	(if (not L) <i>TURN_L</i> <i>MOVE_F</i>)

Table 3. Genotype of typical individual obtained in Case A (Type I)

Table 3 and Fig. 6 show the genotype and phenotype of a typical individual obtained in Case A, respectively. This individual divides the state space into two regions and assigns two actions. Here, we defined this kind of individual as Type I. This type occupies 52.5% out of 200 individuals in Case A and accomplishes the task of maintaining a certain distance from the target by using the following simple strategy. As shown in the intelligence gene of Table 3, if L is not true, then *TURN_L* is executed; in other words, if the left visual sensor does not recognize the target, the robot turns left (State 1 in Fig. 6). Otherwise, if L is true, *MOVE_F* is executed, that is, if the left visual sensor recognizes the target, the robot moves forward (State 2 in Fig. 6). Here, *MOVE_F* in the state space is arranged in the right front of the robot and *TURN_L* occupies the rest of the state space. Further, the robot has two visual sensors, but actually uses only one. In Case A, the robot is two times faster than the target and collides with it if the *MOVE_F* is arranged in front of the robot. Thus, the Type I robot avoids a collision and maintains the objective distance by shifting the *MOVE_F* from the front and rotating frequently.

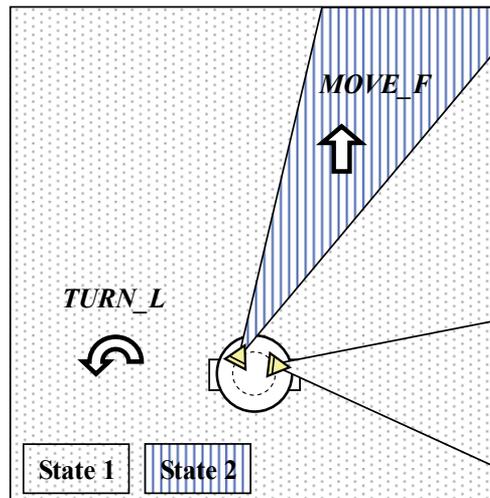


Figure 6. Phenotype of typical individual expressed by the genotype of Table 3 (Type I)

Morphology genes	$\alpha_R=63, \beta_R=27, \gamma_R=82$
[deg]	$\alpha_L=48, \beta_L=50, \gamma_L=30$
Intelligence genes	(if L (if R TURN_L MOVE_F) (if (not R) TURN_R MOVE_B))

Table 4. Genotype of typical individual obtained in Case B (Type II)

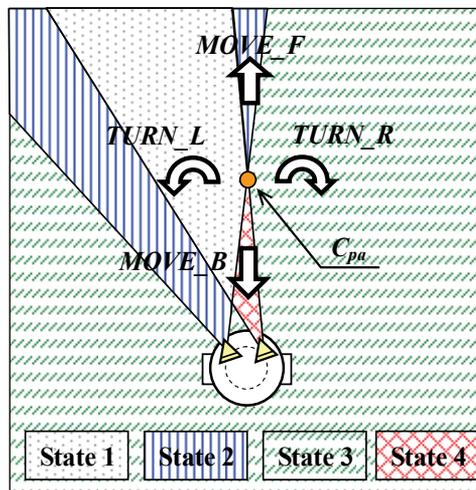


Figure 7. Phenotype of typical individual expressed by the genotype of Table 4 (Type II)

Next, Table 4 and Fig. 7 show the genotype and phenotype of a typical individual obtained in Case B. This individual divides the state space into four regions and assigns four actions. Here, we defined this kind of robot as Type II. This type occupies 57.5% out of 200 individuals in Case B and accomplishes the task by using the following strategy. As shown in the intelligence gene of Table 4, if *L* and *R* are true, then *TURN_L* is executed, that is, if

both sensors recognize the target, the robot turns left (State 1 in Fig. 7). If L is true and R is not true, then $MOVE_F$ is executed; in other words, if the left visual sensor recognizes the target and the right visual sensor does not, the robot moves forward (State 2 in Fig. 7). If both R and L are not true, then $TURN_R$ is executed, that is, if neither the left visual sensor nor the right visual sensor recognizes the target, the robot turns right (State 3 in Fig. 7). If L is not true and R is true, $MOVE_B$ is executed, that is, if the left visual sensor does not recognize the target and the right visual sensor does, the robot moves backward (State 4 in Fig. 7). Here, $MOVE_F$ in the state-action space is arranged in the front of the robot, $TURN_L$ and $TURN_R$ are to the left and right of the $MOVE_F$ region, and $MOVE_B$ is between $MOVE_F$ and the robot. In Case B, the robot is two times slower than the target and needs to approach the target along the shortest path. Therefore, $MOVE_F$ should be arranged in the front of the robot. Additionally, the arrangement of $TURN_L$ and $TURN_R$ for $MOVE_F$ allows a fast search and the centering of the target. Furthermore, when the robot gets too close to the target, it moves backward and maintains a distance between the robot and the target. With this state-action space, Type II obtains better fitness as compared to the others in Case B.

4.4 Discussion: Structural Symmetry Properties

Table 5 shows the averages of the structural indices of the symmetry properties: $|\alpha_L - \alpha_R|$, $|\beta_L - \beta_R|$, and $|\gamma_L - \gamma_R|$, for the elite individuals in the final generation and the standard deviations. Since the standard deviations were high and the averages did not converge, distinguishing structural symmetry properties represented by the arrangement of the two visual sensors were identified. This result shows that a structural symmetry property does not clearly manifest in simulation without considering the physical factor such as sensor weight.

	Case A [deg]		Case B [deg]	
	Ave.	Std. dev.	Ave.	Std. dev.
$ \alpha_L - \alpha_R $	29.5	27.2	25.5	19.4
$ \beta_L - \beta_R $	14.3	7.9	14.5	13.3
$ \gamma_L - \gamma_R $	37.4	29.1	42.4	36.0

Table 5. Results of structural indices obtained in Cases A and B

4.5 Discussion: Functional Symmetry Properties

From the viewpoint of state-action space, we discuss the phenotype of Type II. Figure 8 shows the state-action space of Type II and the physical arrangement of the two visual sensors. As shown by the broken lines, the state-action space of Type II is symmetric about the line between the actual cross-point and the center of the robot, because $MOVE_F$ and $MOVE_B$ are arranged in the front and back for the actual cross-point and $TURN_L$ and $TURN_R$ are to its left and right. In this study, we define such symmetry of the state-action space as functional symmetry. This result shows that from the viewpoint of physical structure, the arrangement of the two visual sensors is not symmetric (the lower area marked by the broken line in Fig. 8), but from the viewpoint of control, the state-action space is symmetric. Table 6 shows the incidence ratio of an individual with functional

symmetry obtained in Cases A and B. Since the ratios are 10.0% in Case A and 57.5% in Case B, the relative velocity difference must be one of factors that generate the functional symmetry. Table 7 shows the average of the actual cross-point angle of the individual with functional symmetry obtained in Cases A and B. Here, if the actual cross-point is 0 [deg] (i.e., exists on the traveling direction line), it means that the state-action space is almost symmetric about the traveling direction. The actual cross-point angle in Case B is lower than that in Case A, that is, the individual obtained in Case B is more symmetric. Furthermore, as shown in Fig. 9, 25% of individuals with functional symmetry in Case A created the actual cross-point within 10 [deg], while the percentage of individual in Case B was 90%. This result suggests that in Case B, most of the robotic systems designed the actual cross-point in front of the robot and assigned actions based on this point. Therefore, the condition in Case B tends to generate functional symmetry about the traveling direction as compared with that in Case A. Furthermore, Fig. 10 shows the relationship between the actual cross-point angle and the fitness in Case B. This result shows that since the correlation is 0.38, the smaller the actual cross-point angle (i.e., the more the functional symmetry), the better the fitness. This is considered to be due to the following reason. In Case B, the robot consumes considerable amount of time in chasing the target because the target velocity is twice that of the robot velocity. Thus, the Type II robot has the fastest approach by creating a region of *MOVE_F* in the travelling direction, as shown Fig. 8. In addition, this type of robot can quickly cope with the random behavior of the target by symmetrically assigning actions based on the actual cross-point. Hence, functional symmetry properties about the traveling direction emerging from the arrangement of the two visual sensors are one of the important design principles in Case B. These result may show that, in nature, since a slower predator, for example, a tiger, compared with a prey must efficiently chase, it almost creates the symmetric stereo-vision.

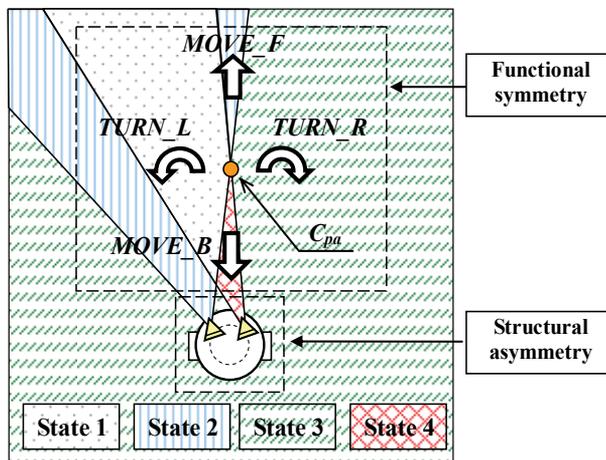


Figure 8. State-action space of individual with functional symmetry

Case A [%]	Case B [%]
10.0	57.5

Table 6. Incidence ratio of individual with functional symmetry obtained in Cases A and B

	Case A [deg]	Case B [deg]
Ave.	12.5	4.6
Std. dev.	3.4	3.7

Table 7. Actual cross-point angle obtained in Cases A and B

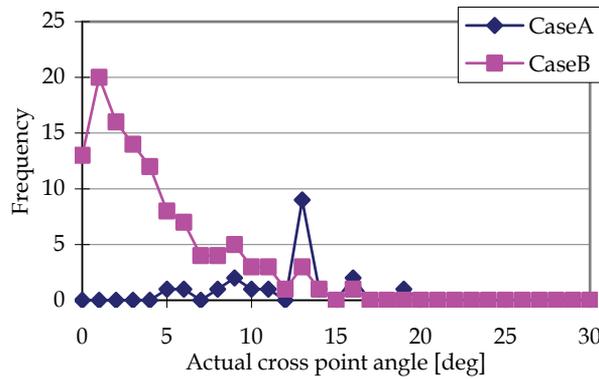


Figure 9. Comparison of actual cross-point angle distribution obtained in Cases A and B

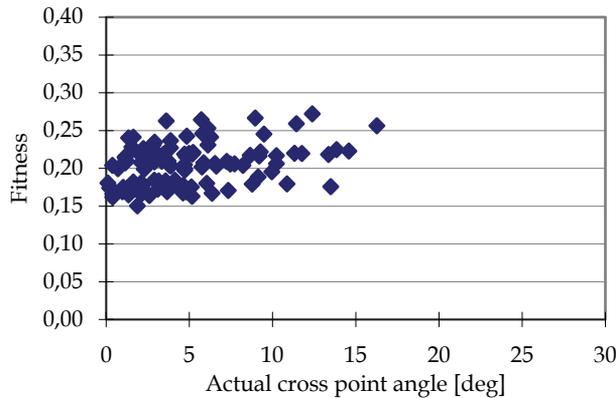


Figure 10. Correlation between actual cross point angle and fitness obtained in Case B

5. Evolutionary Simulation II

5.1 Condition for Simulation

To investigate the influence of genetic noise on the manifestation of symmetry properties, we performed same simulations identical to Evolutionary Simulation I for the noise ratios: 0%, 25%, 50%, 75%, and 100%. Here, the genetic noise is a genotype-phenotype noise (G-P noise) that is added during the transformation process from the genotype to the phenotype. From this, an individual with same genotype is translated to slightly different phenotypes and is given a different fitness value. This G-P noise may be similar to an

acquired disposition in nature. The G-P noise adds a disturbance from -1.0 [deg] to 1.0 [deg] to α , β , and γ of the genotype according to the normal probability distribution. Note that the change in the sensor direction in the traveling direction due to this G-P noise is less than 2.0 [deg], and that the change in the edge of the field of view is less than 2.5 [deg].

5.2 Results and Discussion

Table 8 shows the incidence ratio of an individual with functional symmetry for different G-P noise ratios. From this, we find that in Case A, the incidence ratio of functional symmetry gradually decreases with an increase in the G-P noise, and in Case B, there is a peak in the incidence ratio of functional symmetry depending on the G-P noise ratio. We consider this mechanism as follows. Type I is robust against the G-P noise as compared with Type II. Since Type II designs the state-action space based on the cross-point, a change in the cross-point due to the G-P noise deteriorates the fitness. However, Type I is not affected much, since it does not have a cross-point. Thus, in Case A, Type II is eliminated with an increase in the G-P noise. Consequently, the individuals in Case A lose one visual sensor through evolution and become Type I with high robustness in the presence of G-P noise. The Type I visual sensor has a bias angle (approximately 30 [deg]) for the traveling direction and is asymmetric. Hence, Case A creates functional asymmetry.

On the other hand, Type II must use the state-action space with a cross-point for the fastest chase. Therefore, Type II is not eliminated by increasing the G-P noise in Case B. Moreover, a small G-P noise increases the incidence of functional symmetry. Hence, Case B creates functional symmetry. From this, in this case study, we conclude that Case A, in which the robot is faster, creates functional asymmetry and Case B, in which the target is faster, creates functional symmetry.

G-P Noise	Case A	Case B
0%	10.0	57.5
25%	14.0	57.5
50%	6.0	80.5
75%	5.0	57.5
100%	0.5	56.5

Table 8. Incidence ratios of individual with functional symmetry

6. Conclusions

We focused on symmetry properties and performed computational simulations by using an evolutionary robotic system with reconfigurable morphology and intelligence. First, we investigated the mechanism of emergent symmetry properties for two different relative velocities of the robot and the target and the influence by which the genetic noise gives to the symmetry properties. Although, from the viewpoint of physical structure,

distinguishing structural symmetry properties were identified in simulation without considering the physical factor such as sensor weight, from the viewpoint of control, functional symmetric properties were manifested; functional asymmetry was designed in Case A in which the robot was faster than the target, and functional symmetry was designed in Case B in which the robot was slower than the target. Genotype-phenotype noise, which creates different individuals from the same genotype, improved the robustness of the robot in Case A and raised the incidence ratio of functional asymmetry. On the other hand, a small genotype-phenotype noise improved the incidence ratio of functional symmetry in Case B. In a future study, we intend to investigate the relationship between the sensory system and the driving system using an evolutionary robotic system that is capable of changing not only the sensor arrangement but also the motor arrangement. Additionally, we aim to further investigate the design principles leading to structural symmetry. Furthermore, we will employ physical experiments and attempt to reveal the characteristics of symmetry properties in the real world.

7. References

- Bongard, J. C. & Paul, C. (2000). Investigation Morphological Symmetry and Locomotive Efficiency using Virtual Embodied Evolution, From animaru to animates 6, pp. 420-429
- Bongard, J. C. & Pfeifer, R. (2002). A Method for Isolating Morphological Effects on Evolved Behavior, 7th International Conference on the Simulation of Adaptive Behavior, pp. 305-311
- Hara, F. & Pfeifer, R. (2003). Morpho-functional Machines : The New Species, Springer
- Harvey, I. ; Husband, P. & Cliff, D. (1993). Issues in Evolutionary Robotics, From animals to animates 2, pp. 364-373
- Holland, J. H. (1975). Adaptation in natural and Artificial System, Univ. Of Michigan Press
- Kikuchi, K. & Hara, F. (1998). Evolutionary Design of Morphology and Intelligence in Robotic System Using Genetic Programming, in Proc. Fifth International Conference of the Society for Adaptive Behavior 98, pp. 540-545
- Kikuchi, K. & Hara, F. (2000). A Study on Evolutionary Design in Balancing Morphology and Intelligence of Robotic System, JSME, Journal of Robotics and Mechatronics, pp. 180-189,
- Kikuchi, K.; Hara, F. & Kobayashi, H. (2001). Characteristics of Function Emergence in Evolutionary Robotic Systems -Dependency on Environment and Task-, in Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2288-2293
- Knudsen, E. I. (2002). Instucted learning in the auditory localization pathway of the barn owl, Nature, Vol. 417, pp. 322-328
- Lipson, H. & Pollack, J. B. (2000). Automatic design and manufacture of robotic lifeforms, Naturem, Vol. 406, pp. 974-978
- Pascual, A.; Huang, K.; Neveu, J. & Preat, T. (2004). Brain asymmetry and long-term memory, Nature, Vol. 427
- Pfeifer, R. & Scheier, C. (1999). Understanding Intelligence, The MIT press
- Richard, S. (1989). Temporal Credit Assignment in Reinforcement Learning, PhD thesis, Univ. Cambridge, England

- Salomon, R. & Lichtensteiger, L. (2000). Exploring different Coding Schemes for the Evolution of an Artificial Insect Eye, in Proc. First IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks
- Sims, K. (1994). Evolution 3D Morphology and Behavior by Competition, *Artificial Life 4*, MIT press, pp. 28-39
- Ventrella, J. (1994). Exploration in the Emergence of Morphology and Locomotion Behavior in Animated Characters, *Artificial Life 4*, pp. 436-441

A Quantitative Analysis of Memory Usage for Agent Tasks

DaeEun Kim

*Yonsei University, School of Electrical and Electronic Engineering
Corea (South Korea)*

1. Introduction

Many agent problems in a grid world have been handled to understand agent behaviours in the real world or pursue characteristics of desirable controllers. Normally grid world problems have a set of restricted sensory configurations and motor actions. Memory control architecture is often needed to process the agent behaviours appropriately. Finite state machines and recurrent neural networks were used in the artificial ant problems (Jefferson et al., 1991). Koza (1992) applied genetic programming with a command sequence function to the artificial ant problem. Teller (1994) tested a Tartarus problem by using an indexed memory. Wilson (1994) used a new type of memory-based classifier system for a grid world problem, the Woods problem.

The artificial ant problem is a simple navigation task that imitates ant trail following. In this problem, an agent must follow irregular food trails in the grid world to imitate an ant's foraging behaviour. The trails have a series of turns, gaps, and jumps on the grid and ant agents have one sensor in the front to detect food. Agents have restricted information of the surrounding environment. Yet they are supposed to collect all the food on the trails. The first work, by Jefferson et al. (1991), used the John Muir trail, and another trail, called Santa Fe trail, was studied with genetic programming by Koza (1992). The trails are shown in Fig. 1. This problem was first solved with a genetic algorithm by Jefferson et al. (1991) to test the representation problem of controllers. A large population of artificial ants (65,536) were simulated in the John Muir trail with two different controller schemes, finite state machines and recurrent neural networks. In the John Muir trail, the grid cells on the left edge are wound adjacent to those on the right edge, and the cells on the bottom edge are wound adjacent to those at top. Each ant has a detector to sense the environment and an effector to wander about the environment; one bit of sensory input to detect food and two bits of motor actions to move forward, turn right, turn left and think (no operation). Its fitness was measured by the amount of food it collects in 200 time steps. At each time step, the agent can sense the environment and decide on one of the motor actions. The behaviours of ant agents in the initial population were random walks. Gradually, more food trails were traced by evolved ants. Koza (1992) applied genetic programming to the artificial ant problem with the Santa Fe trail (see Fig. 1(b)), where he assumed it as a slightly more difficult trail than the John Muir trail, because the Santa Fe trail has more gaps and turns between food pellets. In his approach, the control program has a form of S-expression (LISP) including a sequence of actions and conditional statements.

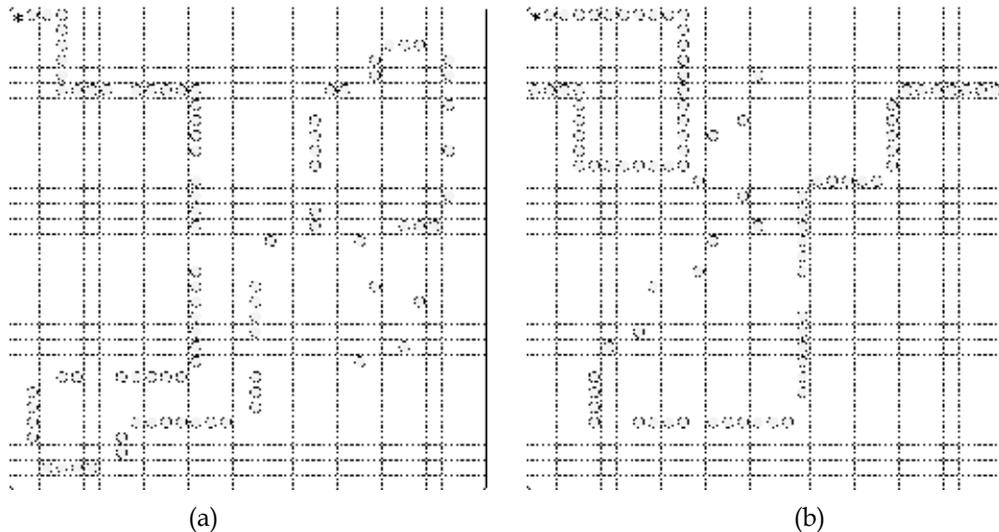


Figure 1. Artificial ant trails (*: ant agent, circle: food) (a) John Muir trail (Jefferson et al., 1991) (b) Santa Fe trail (Koza, 1992)

Many evolutionary approaches related to the artificial ant problem considered the fitness as the amount of food that the ant has eaten within a fixed number of time steps (Jefferson et al., 1991, Koza, 1992, Angeline, 1998, Silva et al., 1999). The approaches suggested a new design of control structures to solve the problem, and showed the best performance that they can achieve. The artificial ant problem is an agent problem that needs internal memory to record the past sensory readings or motor actions. However, there has been little discussion for the intrinsic properties related to memory to solve the problem, although the control structures studied so far have a representation of internal memory.

Internal memory is an essential component in agent problems in a non-Markovian environment (McCallum, 1996, Lanzi, 1998, Kim and Hallam, 2002). Agents often experience a perceptual aliasing problem (when the environmental features are not immediately observable or only partial information about the environment is available to an agent, the agent needs different actions with the same perceived situation) in non-Markovian environment. For instance, in the artificial ant problem an ant agent has two sensor states with one sensor, food detected or not in the front, but it needs different motor actions on the same sensor state, depending on the environmental feature. Thus, a memoryless reactive approach is not a feasible solution for the problem. There have been memory-encoding approaches to solve agent problems or robotic tasks in a non-Markovian environment or partially observable Markov decision process (POMDP). Colombetti and Dorigo (1994) used a classifier system to learn proper sequences of subtasks by maintaining internal state and transition signals which prompt an agent to switch from one subtask to another. Lanzi (2000) has shown that internal memory can be used by adaptive agents with reinforcement learning, when perceptions are aliased. Also there have been researches using a finite-size window of current and past observations and actions (McCallum, 1996, Lin and Mitchell, 1992). Stochastic approaches or reinforcement learning with finite state controllers have been applied to POMDPs (Meuleau et al., 1999, Peshkin et al., 1999, Braziunas and Boutiler, 2004). Bram and de Jong (2000) proposed a means of counting the number of internal states required to perform a particular task in an environment. They estimated state counts from

finite state machine controllers to measure the complexity of agents and environments. They initially trained Elman networks (Elman, 1990) by reinforcement learning and then extracted finite state automata from the recurrent neural networks. As an alternative memory-based controller, a rule-based state machine was applied to robotic tasks to see the memory effect (Kim and Hallam, 2001). Later Kim and Hallam (2002) suggested an evolutionary multiobjective optimization method over finite state machines to estimate the amount of memory needed for a goal-search problem.

Generally, finding an optimal memory encoding with the best behaviour performance in non-Markovian environments is not a trivial problem. Evolutionary computation has been a popular approach to design desirable controllers in agent problems or robotic tasks (Nolfi and Floreano, 2000). To solve the artificial ant problem, we will follow the evolutionary robotics approach. In the evolutionary approach, the behaviour performance of an ant agent is scored as fitness and then the evolutionary search algorithm with crossover and mutation operators tries to find the best control mapping from sensor readings to actions with a given memory structure. Here, we focus on the question of how many memory states are required to solve the artificial ant problem in non-Markovian environment or what is the best performance with each level of memory amount. This issue will be addressed with a statistical analysis of fitness distribution.

An interesting topic in evolutionary computation is to estimate the computational effort (computing time) to achieve a desirable level of performance. Koza (2002) showed a statistic to estimate the amount of computational effort required to solve a given problem with 99% probability. In his equation, the computational effort $I(m,z)$ is estimated as

$$I(m,z) = m g \log(1-z) / \log(1-P(m,g)) \quad (1)$$

where m is the population size, g is the number of generations, z is the confidence level which is often set to 0.99, and $P(m,g)$ is the probability of finding a success within mg evaluations. However, this equation requires to estimate $P(m,g)$ accurately. It has been reported that the measured computational effort has much deviation from the theoretical effort (Christensen and Oppacher, 2002, Niehaus and Banzhaf, 2003). The probability $P(m,g)$ has been measured as the number of successful runs over the total number of runs. In that case, $P(m,g)$ does not consider how many trial runs are executed, which can lead to the inaccurate estimation of the computational effort. The estimation error can be observed, especially when only a small number of experimental runs are available (Niehaus and Banzhaf, 2003). Lee (1998) introduced another measure of the computational effort, the average computing cost needed to obtain the first success, which can be estimated with $mg(\alpha+\beta+2)/(\alpha+1)$ for given α successes and β failures. As an alternative approach to the performance evaluation, the run-time distribution, which is a curve of success rate depending on computational effort, has been studied to analyze the characteristics of a given stochastic local search (Hoos and Stuetzle, 1998, 1999). However, this measure may also experience the estimation error caused by variance of success probability.

Agent problems in a grid world have been tested with a variety of control structures (Koza, 1992, Balakrishnan and Honavar, 1996, Ashlock, 1997, 1998, Lanzi, 1998, Silva et al., 1999, Kim, 2004), but there has been little study to compare control structures. Here, we introduce a method of quantitative comparisons among control structures, based on the behaviour performances. The success rate or computational effort is considered for the purpose. In this paper we use finite state machines as a quantifiable memory structure and a varying

number of memory states will be evolved to see the memory effect. To discriminate the performances of a varying number of memory states, we provide a statistical significance analysis over the fitness samples. We will present a rigorous analysis of success rate and computational effort, using a beta distribution model over fitness samples. Ultimately we can find the confidence interval of the measures and thus determine statistical significance of the performance difference between an arbitrary pair of strategies. This analysis will be applied to show the performance difference among a varying number of internal states, or between different control structures. The approach is distinguished from the conventional significance test based on the *t*-statistic. A preliminary report of the work was published in the paper (Kim, 2006).

We first introduce memory-encoding structures including Koza's genetic programming and finite state machines (section 2) and show methods to evaluate fitness samples collected from evolutionary algorithms (section 3). Then we compare two different control structures, Koza's genetic programming controllers and finite state machines, and also investigate how a varying number of internal states influence the behaviour performance in the ant problem. Their performance differences are measured with statistical significance (section 4).

2. Memory-Encoding Structures

In this section we will show several different control structures which can encode internal memory, especially focusing on genetic programming and finite state machines. The two control structures will be tested in the experiments to quantify the amount of memory needed for the ant problem.

2.1 Genetic Programming approach

Koza (1992) introduced a genetic programming approach to solve the artificial ant problem. The control structure follows an S-expression as shown in Fig. 2. The ant problem has one sensor to look around the environment, and the sensor information is encoded in a conditional statement **if-food-ahead**. The statement has two conditional branches depending on whether or not there is a food ahead. The **progn** function connects an unconditional sequence of steps. For instance, the S-expression (**progn2 left move**) directs the artificial ant to turn left and then move forward in sequence, regardless of sensor readings. The **progn** function in the genetic program corresponds to a sequence of states in a finite automaton.

```
(if-food-ahead (move)
  (progn3 (left)
    (progn2 (if-food-ahead (move) (right))
      (progn2 (right)
        (progn2 (left) (right))))
    (progn2 (if-food-ahead (move) (left))
      move))))
```

Figure 2. Control strategy for Santa Fe trail by Koza's genetic programming (Koza, 1992) ; **(if-food-ahead (move) (right))** means that if food is found ahead, move forward, else turn right, and **progn2** or **progn3** defines a sequence of two actions (subtrees) or three actions (subtrees)

In Koza's approach, a fitness measure for evolving controllers was defined as the amount of food ranging from 0 to 89, traversed within a given time limit. An evolved genetic program did not have any numeric coding, but instead a combination of conditional statements and motor actions were represented in an S-expression tree without any explicit state specification. The evaluation of the S-expression is repeated if there is additional time available. Fig. 2 is one of the best control strategies found (Koza, 1992).

Following Koza's genetic programming approach, we will evolve S-expression controllers for the Santa Fe trail in the experiments. Here, we use only two functions, **if-food-ahead** and **progn2** to restrict evolving trees into binary trees, and **progn3** can be built with the primitive function **progn2**. In the evolutionary experiments, the number of terminal nodes in an S-expression tree will be taken as a control parameter. As a result, we can see the effect of a varying number of leaf nodes, that is, a variable-length sequence of motor actions depending on the input condition. Later we will explain how the control parameter is related with the amount of memory that the S-expression tree uses.

2.2 Finite State Machines

A simple model of memory-based systems is a Boolean circuit with flip/flop delay elements. A Boolean circuit network with internal memory is equivalent to a finite state machine (Kohavi, 1970). Its advantage is to model a memory-based system with a well-defined number of states, and allows us to quantify memory elements by counting the number of states. The incorporation of state information helps an agent to behave better, using past information, than a pure reaction to the current sensor inputs. Finite state machines have been used in evolutionary computation to represent state information (Fogel, 1966, Stanley et al., 1995, Miller, 1996).

A Finite State Machine (FSM) can be considered as a type of Mealy machine model (Kohavi, 1970, Hopcroft and Ullman, 1979), so it is defined as $M = (Q, \Sigma, \Delta, \lambda, q_0)$ where q_0 is an initial state, Q is a finite set of states, Σ is a finite set of input values, Δ is a set of multi-valued outputs, δ is a state transition function from $Q \times \Sigma$ to Q , and λ is a mapping from $Q \times \Sigma$ to Δ , where $\lambda(q, a)$ in Δ . $\delta(q, a)$ is defined as the next state for each state q and input value a , and the output action of machine M for the input sequence $a_1, a_2, a_3, \dots, a_n$, is $\lambda(q_{x0}, a_1), \lambda(q_{x1}, a_2), \lambda(q_{x3}, a_3), \dots, \lambda(q_{x(n-1)}, a_n)$, where $q_{x0}, q_{x1}, q_{x3}, \dots, q_{xn}$ is the sequence of states such that $\delta(q_{xk}, a_{k+1}) = q_{x(k+1)}$ for $k=0, \dots, n-1$.

state	Input 0	Input 1
q_0	q_1, L	q_0, M
q_1	q_2, R	q_2, M
q_2	q_3, R	q_3, R
q_3	q_0, M	q_4, M
q_4	q_0, M	q_0, M

a)

state	Input 0	Input 1
q_0	q_7, R	q_6, M
q_1	q_6, N	q_2, R
q_2	q_5, R	q_5, M
q_3	q_0, R	q_1, L
q_4	q_2, L	q_5, M
q_5	q_6, N	q_4, M
q_6	q_0, R	q_6, M
q_7	q_2, R	q_2, L

b)

state	Input 0	Input 1
q_0	q_5, R	q_6, N
q_1	q_0, R	q_0, N
q_2	q_4, M	q_3, M
q_3	q_6, R	q_3, M
q_4	q_5, M	q_0, L
q_5	q_3, M	q_4, L
q_6	q_1, R	q_2, M

c)

Table 1. Finite state machines for Santa Fe trail problem (a) 405 time steps, with 5 states (b) 379 time steps, with 8 states (c) 356 time steps, with 7 states (input 1: food ahead, input 0: no food ahead), output set is L (turn left), R (turn right), M (move forward), N (no-operation)

FSM can be used as a quantifiable memory structure, but developing an optimal state transition mapping needs an exhaustive search and so we apply a genetic algorithm to find desirable FSM controllers for the agent problem. An FSM controller is denoted as a numeric chromosome consisting of integer strings, unlike Koza's genetic programming. The chromosome represents a state transition table in a canonical order and its initial state is 0 by default. That is, the gene coding is defined here as a sequence of the pair (state number, state output) of each sensor value in canonical order of state number. For example, the gene coding in Table 1(b) is represented as:

7R6M 6N2R 5R5M 0R1L 2L5M 6N4M 0R6M 2R2L

where motor actions will also be coded numerically. A set of sensor configurations is defined for each internal state, following the Mealy machine notation. The encoding of the Mealy machine can easily represent sequential states. However, it needs an encoding of complete sensory configurations for each state and scales badly with growing machine complexity. This control structure is useful to agents with a small number of sensors, since the chromosome size in finite state machines is exponentially proportional to the number of sensors. The artificial ant has one sensor, and the Mealy machine will have a reasonable size of chromosome even for a large number of internal states.

If there is a repeated sequence of actions, $(A_1, A_2, A_3, \dots, A_n)$ to be executed, it can be run by FSM controllers with at most n internal states. Koza's genetic program has a sequence of conditioned actions and so it can be converted into a finite state automaton without difficulty. Terminal nodes in a genetic program of S-expression define motor actions of an ant agent, and the tree traversal by a depth-first-search algorithm relying on sensor readings will guide a sequence of actions. We can simply assign each action in sequence to a separate internal state, and the sequence order will specify the state transition. For example, the function **progn2** or **progn3** has a series of actions, and so the corresponding states defined for the actions will have unconditional, sequential state transitions. The function **if-food-ahead** will have a single internal state for its two actions (the function **if-food-ahead** is supposed to have two children nodes or two subtrees. If an action is not observed immediately at the left or right child, the first terminal node accessed by the tree traversal will be chosen for the internal state.) in the branches, because the actions depending on the sensor reading (input 0 or 1) can be put together in a slot of internal state. With this procedure, we can estimate the number of internal states that a genetic program needs, as the total number of terminal nodes minus the number of **if-food-ahead**'s. The above conversion algorithm will be applied to evolved S-expression trees in the experiments and we can compare the two types of controllers, FSM and S-expression controllers, in terms of memory states. The FSM built with the algorithm may not be of minimal form in some case, because a certain state may be removed if some sequence of actions are redundant, or if nested and consecutive **if-food-ahead**'s appear in the evolved tree (some motor actions for no food may not be used at all). However, the algorithm will mostly produce a compact form of FSMs.

Table 1(a) is a FSM converted from the genetic program shown in Fig. 2; the genetic programming result has a redundant expression (**progn2**) (**left**) (**right**) in the middle of the S-expression and it was not encoded into the FSM. If one looks into the controller in Table 1(a), the behaviour result is almost the same as the Koza's genetic programming result in Fig. 2, even if they are of different formats. Sequential actions were represented as a set of

states in the finite automaton. The result indirectly implies that five internal states are sufficient to collect all 89 pellets of food. We will investigate later what is the minimal amount of memory required to achieve the task. When we evolved FSMs such that ants collect all the food in the grid world, the controllers in Table 1(b)-(c) as well as Table 1(a) were obtained in a small number of generations. The FSM controller in Table 1(a) takes 405 time steps to pick up all 89 pellets of food, while Table 1(b) and Table 1(c) controllers need 379 time steps and 356 time steps, respectively. As an extreme case, a random exploration in the grid would collect all the food if there is a sufficient time available. Thus, the efficiency, that is, the number of time steps needed to collect all the food can be a criterion for better controllers. In this paper, we will consider designing efficient controllers with a given memory structure and explore the relation between performance and the amount of memory.

2.3 Other control structures

Recurrent neural networks have been used to tackle the artificial ant problem (Jefferson et al., 1991, Angeline et al., 1994). The evolved network had a complex interconnectivity and dynamic control process. The attractors and transient states are often hardly identifiable. The amount of memory needed for the problem is not easily quantifiable with the recurrent neural structure, although it can solve the problem through its dynamic process.

$$\begin{aligned}
 A0 &= A2 \times \text{NoFood} \times A_2 \\
 A1 &= A2 + 1.434651 / \text{NoFood} \\
 A2 &= 0.852099 / \text{NoFood} \\
 A3 &= A2 / (-0.198859 + A2 + 0.200719) \times (-0.696450(A1 - A3) / A3)
 \end{aligned}$$

Figure 3. An example of multiple interacting programs; *NoFood* is a sensor and $A0, \dots, A3$ are the actions for no operation, right, left and move, respectively. (Angeline, 1998)

For another memory structure, a concept of multiple interacting programs has been developed to represent state information (Angeline, 1998). A program consists of several symbolic expressions evolved, which can refer to the output of a symbolic equation. A set of symbolic equations corresponds to a set of dynamic state equations, and the program is similar to recurrent neural networks in that the output of one equation can be connected to the inputs of the other equations. Fig. 3. is a program example for the Santa Fe trail problem which has one sensor and four output actions (Angeline, 1998).

The above control structures, recurrent neural networks and multiple interacting programs, have an advantage of representing a complex dynamic operation, but their representations are not quantifiable in terms of internal states. Especially recurrent neural networks have been popularly used in many agent problems with non-Markovian environment (Nolfi and Floreano, 2000), but it would be a difficult task to identify the internal states directly or recognize the relevance and role of internal memory. In contrast, finite state machines and Koza's genetic programming can quantify the amount of internal memory needed for a given agent problem and allow us to analyze the role of internal states on the behaviour performance. In the experiments, we will show this quantitative approach and compare the two different control structures.

3. Evaluation of Fitness Distribution

3.1 *t*-distribution

Student's *t*-test is useful to evaluate the fitness distribution by observing a small number of samples (Cohen, 1995). By calculating the mean and standard deviation over the sample set, the actual mean would be estimated in the range

$$[x - t_{(n, \epsilon/2)} \sigma/\sqrt{n}, x + t_{(n, \epsilon/2)} \sigma/\sqrt{n}]$$

where n is the number of samples, x is the mean of samples, $1-\epsilon$ is a confidence level, and $t_{(n, \epsilon/2)}$ is the coefficient to determine the interval size, which depends on the number of samples, n . The population standard deviation is unknown, so it is estimated from the sample standard deviation σ/\sqrt{n} , where σ is the standard error of samples.

If there are two collections of performance samples over different strategies, we first draw the confidence ranges for the two sets of samples by the above equation. Then we can statistically determine if one strategy is better than the other. The question reduces to the problem of whether the likely ranges of the population mean overlap. If the ranges overlap each other, we cannot reject the null hypothesis that there is no difference between the two distributions. If the two intervals do not overlap, there is a $(1-\epsilon)100\%$ chance that the two means come from different populations and the null hypothesis should be rejected.

The *t*-test is commonly used to determine confidence intervals or critical region for hypothesis tests. Evolutionary approaches are stochastic and the performance results are often demonstrated with the average performance or confidence intervals by the *t*-test. However, the fitness samples in evolutionary computation rarely follow a normal distribution. The above estimation of the population mean may deviate from the actual performance. A set of samples including many outliers tend to inflate the variance and depress the mean difference between a pair of groups as well as the corresponding statistical significance of the *t*-statistic (Miller, 1986). Thus, we suggest another performance measure based on beta distribution.

3.2 Wilcoxon rank-sum test

The Wilcoxon rank-sum test is a non-parametric comparison over two sets of samples and it is based on the rank of the samples (Wild and Seber, 1999). If there are n_a and n_b observations available for the two groups, respectively, the test first ranks the $n_a + n_b$ observations of the combined sample. Each observation is given its own rank in ascending order, that is, starting with 1 for the smallest value. If there is any tie score of observations, the observations are assigned the average rank. The Wilcoxon rank-sum test calculates, r_s , the sum of the ranks for observations which belong to one sample group. When both n_a and n_b are larger than 10, we can assume the distribution of r_s follows a normal distribution. Thus, we can make a significance test over the difference of the two sample distributions by calculating the probability,

$$\Pr(X > r_s) = \Pr(Z > (w_A - \mu_A)/\sigma_A), \quad \text{or} \quad \Pr(X < r_s) = \Pr(Z < (w_A - \mu_A)/\sigma_A)$$

where $\mu_A = n_a / (n_a + n_b + 1)/2$, $\sigma_A = (n_a n_b (n_a + n_b + 1)/12)^{1/2}$, X is the distribution of a rank-sum, and Z indicates the normal distribution. If the above probability is less than $\epsilon/2$, then we can say that a group of samples is larger than the other with confidence level $(1-\epsilon)$. It rejects the null hypothesis that there is no difference between the two samples.

This Wilcoxon test can be applied to any distribution of samples, regardless of whether it is normal or not, and as well it is not much sensitive to outliers.

3.3 Beta distribution model

The t -test provides a possible range of the average performance for a given strategy or control structure. If there exists a decision threshold to evaluate the performance, we can score a trial run as success or failure. In this case we can consider the success rate as a performance measure. Estimation of success rate can be achieved by empirical data, that is, we can take a finite number of trial runs, and count the number of successful runs. Then the relative frequency of success can be an estimate of the success rate. However, this measure does not reflect the total number of runs. The estimated rate may have large deviation from the real success rate when the number of runs is small. For instance, one successful run out of four trial runs should have a different analysis and meaning with 10 successful runs out of 40 runs, although the relative frequency is the same. Thus, we will explore how to estimate the success rate more accurately.

When it is assumed that $\alpha+\beta$ experiments experience α successes and β failures, the distribution of success rate p can be approximated with a beta distribution. The probability distribution of success rate can be obtained with Bayesian estimation (Ross, 2000), which is different from the maximum likelihood estimation of $p = \alpha / (\alpha + \beta)$. The beta probability density function for the success rate is given by

$$f(p, \alpha, \beta) = p^\alpha (1-p)^\beta / B(\alpha+1, \beta+1)$$

where $B(\alpha+1, \beta+1) = \Gamma(\alpha+1) \Gamma(\beta+1) / \Gamma(\alpha+\beta+2)$ and $\Gamma(n+1) = n\Gamma(n)$.

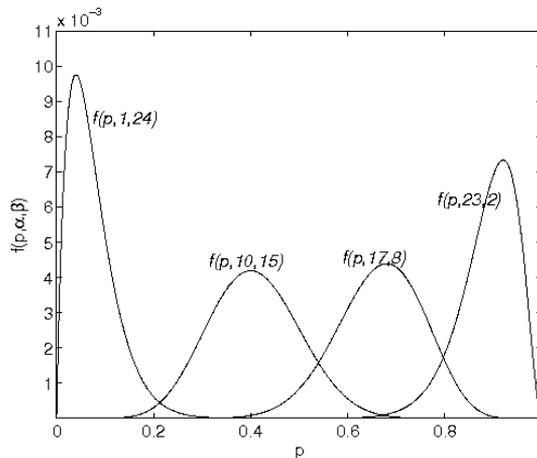


Figure 4. Probabilistic density functions of beta distribution (degree: 25)

Now we want to define confidence intervals about success/failure tests for a given strategy. If the number of experiments is large and the success rate is not an extreme value, then the distribution will be close to a normal distribution. For a small number of samples, the distribution of success probability will follow a beta distribution, not a Gaussian distribution. Fig. 4 shows the probability density function $f(p, \alpha, \beta)$ over a different number of success cases among 25 runs. The beta has very diverse distributions depending on the success cases. For a small or large number of successes, the distribution is far away from

standard normal distribution. Thus, the following numeric approach will be used to estimate confidence intervals. Assume that a random variable X with a beta distribution has the upper bound b_u and the lower bound b_l for confidence limits such that $P(b_l < X < b_u) = 1 - \varepsilon$ and $P(X < b_l) = \varepsilon / 2$. Then we can assert that $[b_l, b_u]$ is a $(1 - \varepsilon)$ 100% confidence interval. If a success probability p is beta-distributed, the confidence limits b_l, b_u can be obtained by solving the following equations:

$$\frac{\varepsilon}{2} = \int_0^{b_u} \frac{p^\alpha(1-p)^\beta}{B(\alpha+1, \beta+1)} dp, \quad \frac{\varepsilon}{2} = \int_{b_l}^1 \frac{p^\alpha(1-p)^\beta}{B(\alpha+1, \beta+1)} dp \quad (2)$$

The lower and upper bound probability b_l, b_u will decide the 95% confidence interval $[b_l, b_u]$. The above method estimates the success rate with confidence interval, which will be more accurate than the maximum likelihood estimation $p = \alpha / (\alpha + \beta)$. It can also provide better estimation of the computational effort given in Equation (1). By using the interval $[b_l, b_u]$ for $P(m, g)$, the 95% confidence interval of the computational effort will be

$$I(m, z) \in \left[C \cdot \frac{\log(1-z)}{\log(1-b_u)}, C \cdot \frac{\log(1-z)}{\log(1-b_l)} \right]$$

where C is the computing cost for a single run, proportional to mg , and $z=0.95$. If the number of runs to estimate the success rate is small, the confidence interval of the success rate becomes large and so the interval size of the computational effort will increase. The confidence interval of the computational effort provides more precise and meaningful information than a single value estimate that Koza (1992) showed.

Now we compute the computational effort in another respect. We assume that a given experiment is repeatedly run until a successful controller is found. A better strategy or methodology will have a smaller number of runs to obtain a successful controller. Thus, the computational effort (computing time) needed to obtain the first success can be a criterion for performance evaluation. The effort test was suggested by Lee (1998) to compare the performance of different strategies for evolutionary experiments, and he used a measure of the average computing cost needed for the first successful run. In this paper the measure will be extended more rigorously to show the confidence interval of the effort cost.

For a given success rate p over the controller test, the average number of trial runs before the first success run can be calculated as

$$E(X) = \sum_{x=1}^{\infty} xp(1-p)^{x-1} = \frac{1-p}{p}$$

where x is the number of trials before the first success. Therefore, the computational effort applied before the first success will be $C(1-p)/p$ where C is a unit computing cost per run (we assume the computational effort only consists of experimental runs who result in failure. If we include the first successful run in the effort, the effort can be estimated with $C_f(1-p)/p + C_s$ where C_f is a failure computing cost and C_s is a success computing cost). If the computing cost per run is variable, we take the averaged cost over multiple runs for an approximate estimation of C . A random variable Y is defined as the amount of trial cost with a success probability p . The expected value of the cost for the random variable Y will be as follows:

$$\begin{aligned}
 E(Y) &= \int_0^1 \frac{1-p}{p} C \cdot \frac{1}{B(\alpha+1, \beta+1)} p^\alpha (1-p)^\beta dp \\
 &= \frac{\alpha + \beta + 1}{\alpha} C - C = \frac{\beta + 1}{\alpha} C
 \end{aligned}$$

If a success rate p has the lower and upper bound probability b_l, b_u by the estimation of confidence interval in Equation (2), the 95% confidence effort cost will be estimated with $[C(1-b_u)/b_u, C(1-b_l)/b_l]$. It is of note that this computational effort is distinguished from Koza (1992)'s estimation given in Equation (1). He assumed that an indefinite number of successful runs, but at least once, are attained with his computational effort, while Lee (1998)'s effort measures the computing cost needed to obtain the first success.

Now we show an example of estimating success rate or computational effort to help understanding of the beta test. When 10 experimental runs are tested for a given task, assume that 9 successes and 1 failure happen for a given strategy. The 95% confidence interval of success rate can be estimated with the two integral equations in Equation (2) ($\epsilon=0.05$). If we solve the equations in terms of b_l, b_u , then

$$\frac{0.05}{2} = \int_0^{b_l} \frac{p^9(1-p)^1}{B(9+1, 1+1)} dp, \quad \frac{0.05}{2} = \int_{b_u}^1 \frac{p^9(1-p)^1}{B(9+1, 1+1)} dp$$

which gives an approximate solution $b_l = 0.587, b_u = 0.977$. Thus, $[0.587, 0.977]$ is a confidence interval for a success rate p in this example. The effort cost interval (Lee's effort cost) will be $[C(1-0.977)/0.977, C(1-0.587)/0.587] = [.02C, .70C]$ and its mean $C(\beta+1)/\alpha$ is $0.22C$. It is expected in the future experiment that 0.02-0.70 times the computing time of a single run will be spent before the first success with 95% confidence. Koza's computational effort will have a confidence range,

$$I(m, g) \in [C \cdot \frac{\log(1-0.95)}{\log(1-0.977)}, C \cdot \frac{\log(1-0.95)}{\log(1-0.587)}] = [0.79C, 3.38C],$$

unlike his estimate of $C \log(1-0.95) / \log(1-0.90)=1.30C$ with the maximum likelihood probability $p=0.90$. If there are 90 successful runs out of 100 trials, the success rate will be $[0.825, 0.944]$ and then $I(m, g)$ is in the range of $[1.04C, 1.72C]$. It shows that we can produce more precise estimation of the effort with more trials.

So far we have shown how to estimate success rate or computational effort related to success rate. By observing a set of fitness samples, we can calculate the confidence interval of success rate, and more experimental runs make the confidence interval of success rate smaller, which means more reliable information of the success rate. To compare strategies, or determine which strategy is more efficient, the confidence interval of success rate or effort cost can be measured for each strategy. Then we can determine statistical significance of the performance difference over strategies. We have shown two kinds of computational efforts which have different meaning of the effort. Both of them depend on the success rate estimation and so they will produce the same decision over the hypothesis test of the performance difference between a pair of groups. In our experiments, the effort cost calculation will use the computing cost needed to obtain the first success, because the computing effort for the first success has a lower level of computing cost and the confidence range is more tight.

4. Experiments

Our evolutionary algorithm will focus on quantifying the amount of memory needed to solve the artificial ant problem. In the ant problem, the fitness function F is defined as follows:

$$F = t_A - \alpha Q_{food}$$

where t_A is the number of time steps required to find all the food, or the maximum allowed amount of time if the ant cannot eat all of them. In the experiments 400 time steps are assigned for each ant agent's exploration, and Q_{food} is the amount of food the ant has eaten. α is a scaling coefficient, which is set to 2 (an ant agent needs 2 time steps on the average to take a food pellet in the nearest neighbours and the coefficient was set to 2). This fitness function considers finding the minimum amount of time to traverse all the food cells. Thus, smaller fitness means better controller. We will use the Santa Fe trail for the target environment. An ant agent may need varying computing time for its fitness evaluation, because t_A varies depending on the time to collect all the food. When an ant succeeds in collecting all the food before 400 time steps, the exploration process can instantly stop not to wait for the whole 400 time steps to complete. This will influence the computing cost and so the computing cost C for a single run will be measured by the averaged CPU run-time over multiple runs.

In this paper, we will test two types of evolving control structures, FSMs and S-expression trees, and compare the performances of ant agents for a given level of memory amount. We will evolve each control structure with a varying number of internal states and analyze fitness samples collected from the evolutionary algorithms.

For the first set of experiments with FSM controllers, the chromosome of a FSM controller is represented as an integer string as described in section 2. One crossover point is allowed only among integer loci, and the crossover rate is given to each individual chromosome, while the mutation rate is applied to every single locus in the chromosome. The mutation will change one integer value into a new random integer value. We used a tournament-based selection method of group size four. A population is subdivided into a set of groups and members in each group are randomly chosen among the population. In each group of four members, the two best chromosomes are first selected in a group and then they breed themselves (more than two chromosomes may have tie rank scores and in this case chromosomes will be randomly selected among the individuals). A crossover over a copy of two best chromosomes, followed by a mutation operator, will produce two new offspring. These new offspring replace the two worst chromosomes in a group. In the experiments, the crossover rate is set to 0.6 and the mutation rate, 0.1, 0.05, or 2 over chromosome length, is applied (the above tournament selection takes a half of the population for elitism, and so the high mutation rate will give more chance of diversity to a new population).

For another memory-based controller, the chromosome of a genetic program is defined as an S-expression tree. The crossover operator on the program is defined as swapping subtrees of two parents. There are four mutation operators available for one chromosome. The first operator deletes a subtree and creates a new random subtree. The subtree to be replaced will be randomly selected in the tree. The second operator randomly chooses a node in the tree and then changes the function (`if-food-ahead` or `progn2`) or the motor action. This keeps the parent tree and modifies only one node. The third operator selects a branch of a subtree and reduces it into a leaf node with a random motor action. It will have the effect of removing redundant subtrees. The fourth operator chooses a leaf node and then

splits it into two nodes. This will assist incremental evolution by adding a function with two action nodes. In the initialization of a population or the recombination of trees, there is a limit for the tree size. The maximally allowable depth of trees is 6 in the initialization, but there is no restriction of the depth after the recombination. The minimum number of leaf nodes is 1 and the maximum number of leaf nodes will be set up as a control parameter in the experiment. If the number of leaf nodes in a new tree exceeds the limit, the tree is mutated until the limit condition is satisfied. The above tournament-based selection of group size four will also be applied to the genetic programming approach. The crossover rate is set to 0.6 and the mutation rate is 0.2.

To quantify the amount of memory, a varying number of states, ranging from 1 state to 20 states, are applied to the artificial ant problem. For each different size of state machines, 50 independent runs with a population size of 100 and 10,000 generations are repeated and their fitness distribution is used for the analysis of memory states. Similar evolutionary experiments are applied to the S-expression controllers with a population size of 500 and 2,000 generations (this parameter setting showed good performance within the limit of 5×10^5 evaluations. This may be due to the fact that genetic programming tends to develop new good offspring through crossover of individuals in a large sized population rather than with the mutation operator (Nolfi and Floreano, 2000)). To compare S-expression and FSM controllers, evolved S-expression controllers are converted into FSMs using the algorithm described in section 2.

4.1 *t*-test for memory analysis

One of the main interests is how many memory states are required for ants to traverse all the food cells. This should also be addressed together with the question if a given time limit of exploration may influence the number of memory states, that is, how the exploration efficiency is related to the amount of memory. Identifying the characteristics of memory structure will be based on empirical data, that is, fitness samples.

To find out an appropriate number of states needed to reach a given performance level, experiments with a fixed number of states is repeated 50 times. We first applied a mutation rate of 0.1 to the evolution of FSMs. Figure 5(a) shows the average fitness result for each number of memory states. The error bars of 95% confidence intervals are displayed with the average performance over 50 runs by assuming a *t*-distribution. The experiment clearly distinguishes the performances of memory states ranging from one to nine states, although more than nine states were not significantly different from nine states. The error bars for a few number of states are very small, because they have consistently poor performances with too few memory units. Another experiment with a mutation rate of 0.05 produced a slightly different result. The performance comparison by *t* statistic shows that memory states from one to seven had a significant difference of performance. However, there was no significant distinction between seven and eight states or between eight and nine states. We note that for each level of memory amount, the average performance with a mutation rate of 0.1 was better than that with a mutation rate of 0.05, but the best performance or the median with a mutation rate of 0.1 was worse for more than eight states. The mutation rate 0.1 tends to advance a more random search in its evolutionary computation, which was not helpful to find small penalty fitness. In contrast, the mutation rate 0.05 produced better control solutions, although the fitness samples had large variance. Thus, the comparison of the average performance by the *t*-test may lose some information behind the fitness distribution.

We also tested a mutation rate of 2 over chromosome length to give similar chances of mutations to each chromosome. The average performance or best performance for each number of states is mostly better than that with a mutation rate of 0.05 or 0.1. Memory states from one to eight states are distinguishable in performance by *t*-statistic. Still a large number of states have no significant difference in their behaviour performance.

Each number of memory states has its own fitness curve with 95% confidence intervals and we can check whether the fitness samples can reach a given threshold of desirable performance. For example, fitness 222 (= 400 - 89 × 2) or below should be obtained to collect all 89 food pellets. The curves in Figure 5 suggest that four memory states or less are inadequate for the artificial ant problem. Also memoryless reactive controllers fail to cover all the food. We can see in the figure that at least five states are needed to cover all 89 food pellets. Koza's genetic programming result shown in Fig. 2, which used five memory states if a redundant expression is removed, is consistent with the analysis of memory requirements by FSM experiments.

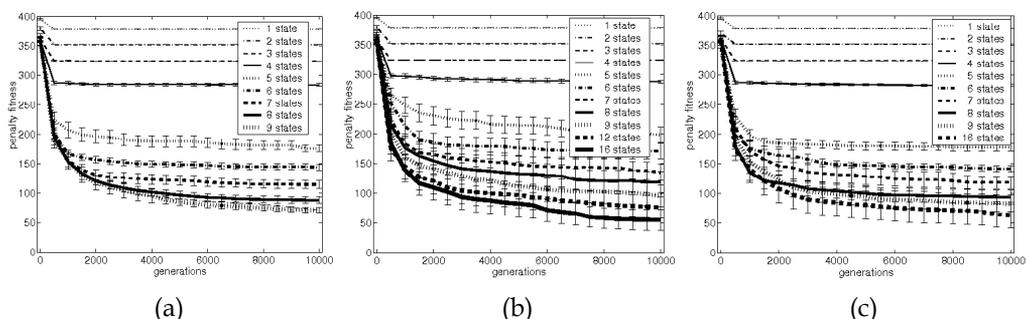


Figure 5. Fitness test experiments in the Santa Fe trail problem (a) a mutation rate of 0.1 (b) a mutation rate of 0.05 (c) a mutation rate of 2 over chromosome length

We still have a question of how many memory states are required if a time limit is given for ants' successful traversals, or how efficiently ants can collect all 89 food pellets. One can guess that the shorter time limit will lead to more memory states to record environmental states. In Figure 5(a), a low penalty fitness 100 (278 time steps for time limit) can be rarely reached with less than seven states, while more than four states succeeded to cover all the food on the grid. The number of encoded states in FSMs differentiates the fitness performance. Better fitness implies that ants explore the trail environment more efficiently and they need less time to collect all the food. Thus, more states have an ability to explore the environment within a shorter time limit. They memorize the environmental features and trigger the next actions promptly by sequentializing a series of actions. The best evolved controller was a 173 time step controller with fitness -5 (= 173 - 89 × 2) as shown in Figure 6. This is comparable to the optimal performance of 165 time steps, in which case the controller memorizes the entire trail completely.

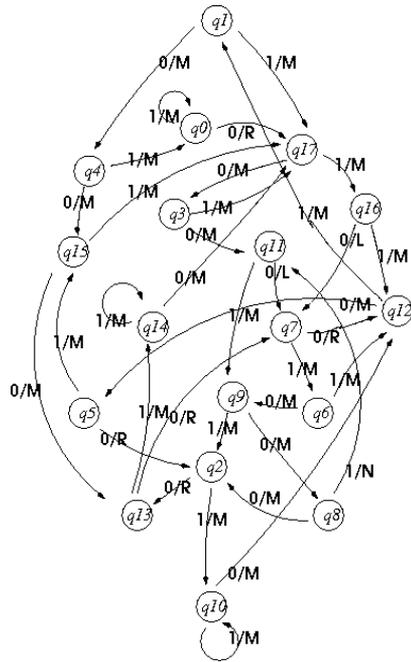
A strategy for an ant agent to collect all the food with five internal states was to look around the environment and move forward if there is no food in the surrounding neighbours. If food is found in one of neighbour cells, the ant immediately moves forward in that direction. The plan needs internal states to take a sequence of actions (*left, right, right, left, move*) or another sequence (*right, left, left, right, move*) with a separate state for one action. If more internal states are given, ants start to utilize environmental features to reduce exploration time, for instance, ants can take three consecutive move actions without looking

around the neighbours after a right or left turn finds food. With a large number of states, more elaborated turns and moves are found in the behaviour of ant agents.

The memory requirement for the task may be determined by the average fitness of multiple runs and its confidence interval, that is, the *t*-statistic (Kim and Hallam, 2001). However, our experiments with a mutation rate of 0.05 or 2 over chromosome length show that the fitness performance has a relatively large variance (error bar) and so the mean difference between a pair of groups is diminished, which influences the statistical significance. Langdon and Poli (1998) showed that the artificial ant problem is highly deceptive, since its fitness landscape is rugged with plateaus, valleys and many local optima. Thus, we suggest a measure of success rate to evaluate fitness samples, based on the beta distribution. This statistic test can be applied to any type of fitness samples regardless of whether they are normally or skewedly distributed.

state	Input 0	Input 1
q0	q17,R	q0,M
q1	q4,M	q17,M
q2	q13,R	q10,M
q3	q11,M	q17,M
q4	q15,M	q0,M
q5	q2,R	q15,M
q6	q9,M	q12,M
q7	q12,R	q6,M
q8	q2,M	q11,N
q9	q8,M	q2,M
q10	q12,M	q10,M
q11	q7,L	q9,M
q12	q5,M	q1,M
q13	q7,R	q14,M
q14	q17,M	q14,M
q15	q13,M	q17,M
q16	q7,L	q12,M
q17	q3,M	q16,M

a)



b)

Figure 6. One of the best evolved control structure FSM in the Santa Fe trail problem (a) transition table (b) state diagram (input 1: food-ahead, input 0: no food ahead, output set is L (turn left), R(turn right), M (move forward), N (no-operation))

4.2 Wilcoxon rank-sum test for memory analysis

For experiments with a mutation rate of 2 over chromosome length, internal states from one to eight generate distinctive difference of performance and they can be discriminated by the *t*-test as shown in Fig. 5, but internal states from eight to ten or more than ten states are not significantly different. Their confidence interval sizes are relatively large and the intervals

overlap, since the fitness distribution has a large variance as shown in Fig. 7; it should be noted that the fitness samples do not follow a Gaussian distribution and have several outliers. By the deceptive property of fitness, it will be difficult to have normal distribution even with a large number of samples. Thus, the Wilcoxon rank-sum test is applied to the fitness samples for any pair of state machines. The best penalty fitness of four-state machines among 25 trials was 280, and the best fitness of five-state machines ranges from 154 to 274. The four-state and five-state machines were serially positioned in ranking order of fitness samples. The Wilcoxon rank-sum test reveals significance in difference of the two samples. In a similar way, we can estimate the partial order of FSMs with a variety number of states. Figure 8 shows the results with a probability p for each pair of machines. The probability $1-p$ represents the confidence level for the partial order relation. We assume 95% confidence level for the relation and if $1-p$ is less than 0.05, we reject the null hypothesis that there is no performance difference between a pair of two sample groups.

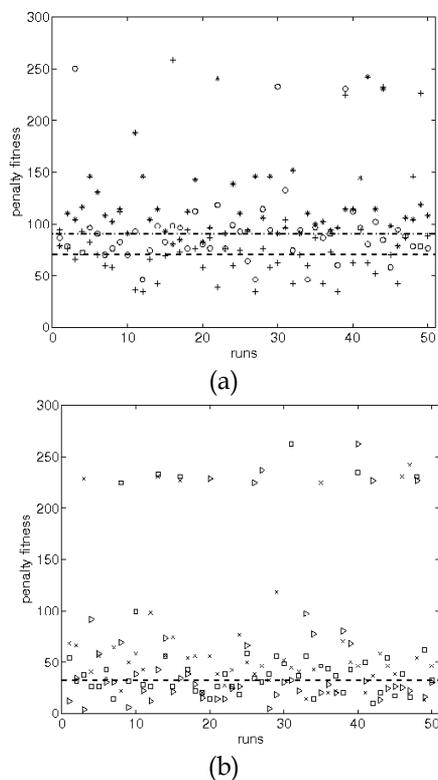


Figure 7. Fitness distribution of FSM controllers (a) with 7 states, 8 states and 9 states (*: 7 states, circle: 8 states, +: 9 states, dotdashed: threshold fitness 90 for success, dashed: threshold fitness 70 for success) (b) with 11 states, 13 states and 15 states (x: 11 states, square: 13 states, triangle: 15 states, dotdashed: fitness 38)

FSMs ranging from one state to 10 states show significant difference among their behaviour performances. The memory effect on the behaviour performance is rather reduced for more than 10 states. For example, a significance level for the difference between 10 states and 11 states or between 11 states and 12 states is not sufficient to reject the null hypothesis. Generally the Wilcoxon rank-sum test show more precision of partial ordering than t -statistic

loses, since it is not sensitive to many outliers. We compared the results with 50 trials and 25 trials for each number of state machines. The 50 trials and 25 trials produced almost the same partial order relations except for the FSMs ranging from 9 to 11 states. The statistic probability p varies depending on the number of trials.

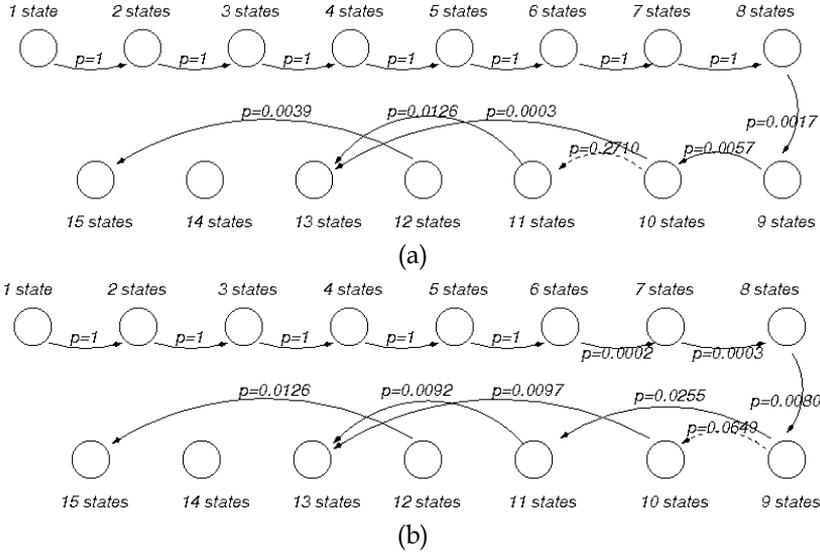


Figure 8. A partial order relation of FSMs with varying number of memory states in terms of behaviour performance by Wilcoxon rank-sum test (a) 50 trials (b) 25 trials (a mutation rate of 2 over chromosome length, solid: significant difference in the Wilcoxon test, dashed: insignificant difference)

4.2 Beta distribution for memory

Now we apply the beta distribution analysis to the fitness samples for finite state machines with a variety of internal states. We assume that if there exists any decision threshold of fitness for success/failure for a given pair of controllers such that it causes significant difference of performance, then the control structures are distinguishable in performance. Between seven and eight states, we place a decision threshold of fitness 90 to determine their success/failure. Then we obtain 5 successes among 50 trials for seven internal states and 27 successes for eight states -- see Figure 7. If we calculate the confidence interval of success rate in beta distribution, seven states and eight states have success rates, $[0.044, 0.214]$ and $[0.403, 0.671]$, respectively, and their effort costs will be $[C_7(1-0.214)/0.214, C_7(1-0.044)/0.044] = [3.67 C_7, 21.73 C_7]$ and $[C_8(1-0.671)/0.671, C_8(1-0.403)/0.403] = [0.49C_8, 1.48C_8]$, where C_7, C_8 is the average computing cost for a single run in failure mode, that is, an experimental run which does not reach the fitness 90. Here, it is assumed that the cost C_7, C_8 is almost identical for the two types of state machines. Then the confidence intervals are significantly different and thus we can clearly see the performance difference of the two control structures. Similarly with a decision threshold of fitness 70, we have 6 successes and 22 successes among 50 trials for eight states and nine states, respectively. Their confidence intervals of the success rate are $[0.057, 0.239]$ and $[0.311, 0.578]$, and their effort costs are $[3.19C, 16.54C], [0.73C, 2.21C]$ by assuming that the two types of state machines have the same computing cost. Then the difference of the success rate or effort cost between the two control structures is also significant.

In fact, the computing cost for each run is variable by different exploration time. In the evolutionary experiments the averaged CPU run-time over multiple runs resulting in failure was $C_{f7} = 39.69$ sec, $C_{f8} = 41.96$ sec for seven and eight states, respectively, while the averaged CPU run-time for success was $C_{s7} = 38.20$ sec, $C_{s8} = 38.36$ sec. The difference of the computing costs does not influence the above significance analysis, because the ratio of the costs is close to 1. In the experiment, the average computing cost for a different size of state machines had a similar level. The decision of significance test by the effort cost agreed with that by the success rate.

By the analysis on the experiments with a mutation rate of 2 over chromosome length, we found that finite states ranging from one to ten have distinctive difference in performance. There was no significant difference between 10 states and 11 states, or between 11 states and 12 states. However, FSMs with 13 states showed significantly better performance than FSMs with 10 states. Fig. 9(a) shows a partial order relation of FSMs with varying number of states, which was estimated by the confidence interval of success rate or effort cost. Evolving more than 15 states produced slightly better controllers than evolving 15 states, but the difference among the fitness samples was not significant. In the same procedure of memory analysis, we tested the statistical significance with a smaller number of trials, 25 trials. We could still obtain similar partial order relation among a varying number of states, although the performance difference between 12 states and 15 states became insignificant. It implies that the beta distribution model can be used for performance comparison even with a small number of trials. The above results show that the beta distribution analysis can find significance of performance differences where fitness samples have a large variance due to outliers or where *t*-statistic does not produce useful information. It is noteworthy that the partial order relation result by the beta distribution is exactly the same as that by Wilcoxon test with 50 trials. The result indirectly supports the validity and usefulness of the beta distribution.

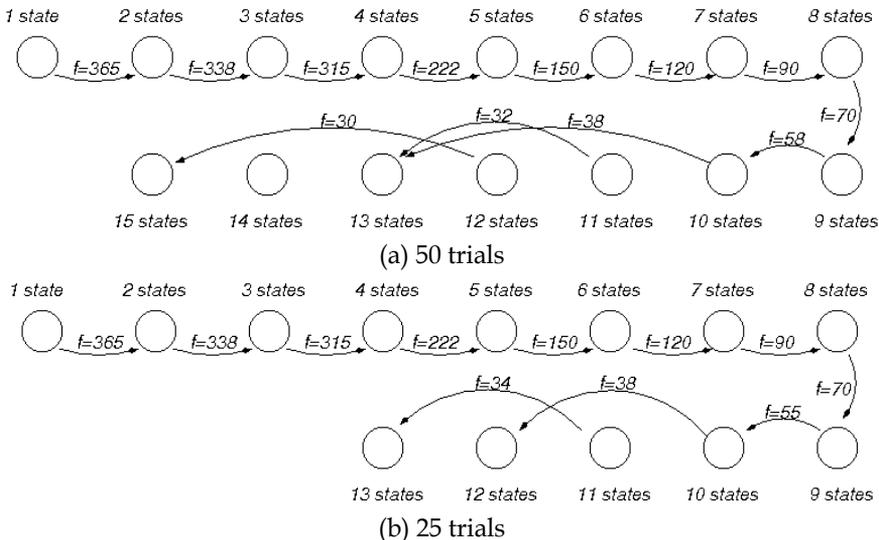


Figure 9. A partial order relation of FSMs with varying number of memory states in terms of behaviour performance by beta distribution (a) 50 trials (b) 25 trials (a mutation rate of 2 over chromosome length is applied and the arrow label indicates the threshold fitness for success/failure)

4.3 Genetic programming approach

For another type of memory-based controllers, we used S-expression controllers. Let n_t , n_f , n_s the number of terminal nodes, the number of the function **if-food-ahead**'s, and the number of internal states in an S-expression tree, respectively. By the conversion algorithm described in section 2, we can build an FSM such that the number of states is $n_s = n_t - n_f$. Fig. 10 shows a conversion example for an evolved S-expression tree. The states $\{q_0, q_1, q_5, q_6\}$ in Fig. 10(b) have different state transitions or motor actions on the input condition and they correspond to the operation of the function **if-food-ahead**. The other states define the same transition and action on the input 0 and 1, which is a copy operation of **progn2**. The conversion algorithm produces a compact form of FSM and helps tracing internal states that the genetic program uses. In this paper we use two control parameters, n_t , n_s to evolve S-expression trees. Once a control parameter is set up for the evolutionary experiments, for instance, the number of terminal nodes is defined, the size of evolved trees should be within the limit size. If an evolved tree is over the limit, the tree should be re-generated by mutation to satisfy the condition.

(if-food-ahead

(move)

(progn2 (right)

(progn2

(if-food-ahead

(progn2

(progn2 (move) (move))

(move))

(progn2

(progn2 (left) (left))

(if-food-ahead

(move) (right))))

(if-food-ahead

(progn2 (move) (move))

(move))))))

a)

state	Input 0	Input 1
q_0	q_1, R	q_0, M
q_1	q_4, L	q_2, M
q_2	q_3, M	q_3, M
q_3	q_6, M	q_6, M
q_4	q_5, L	q_5, L
q_5	q_6, R	q_6, M
q_6	q_0, M	q_7, M
q_7	q_0, M	q_0, M

b)

Figure 10. An example of genetic programming result (a) evolved S-expression ($n_t=12, n_f=4$) (b) converted FSM

In the first experiment, a varying number of terminal nodes were tested by using the number of terminal nodes as a control parameter. A large number of terminal nodes can expect more sequence of actions and thus produce better performance. Figure 11(a) shows the average fitness performance with its 95% confidence range by *t*-statistic for a given number of terminal nodes. Evolving 20 terminal nodes and 30 terminal nodes had 3, 22 cases to reach the fitness 90 or below, respectively, which are similar to the performance of 7 states (5 successes) and 8 states (27 successes) among the FSM controllers in Fig. 5. However, it is not easy to compare directly the performances of FSM and S-expression controllers, since they should have the same criterion for comparison. The best evolved tree for each run often follows the rule that the number of terminal nodes can be approximated by 1.5 times the number of states (see Fig. 12), when the tree is converted into the corresponding FSM structure and the internal states are counted. The best trees with 20 terminal nodes had a range of 9-15 states, and the performance was significantly lower than the performances of

FSM controllers evolved with the same range of states. It indirectly entails that evolving FSMs can provide a better solution to encode internal memory.

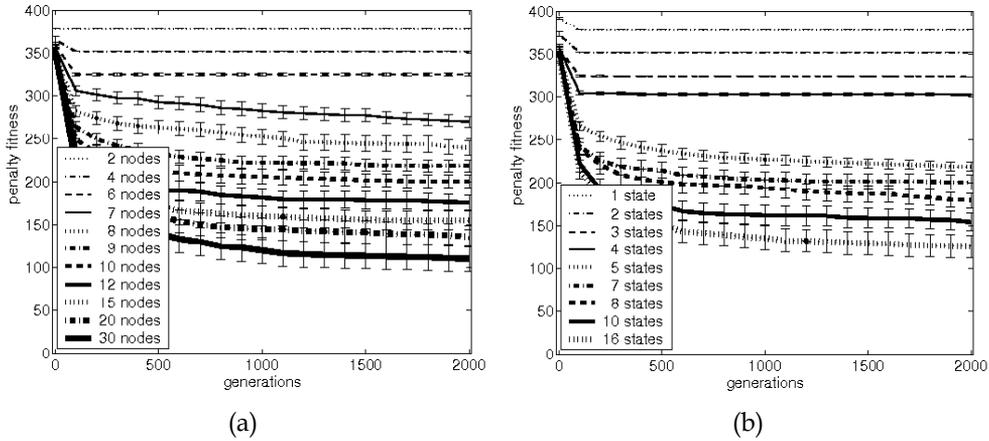


Figure 11. Genetic programming result (a) evolve controllers with a fixed number of terminal nodes (b) evolve controllers with a fixed number of states

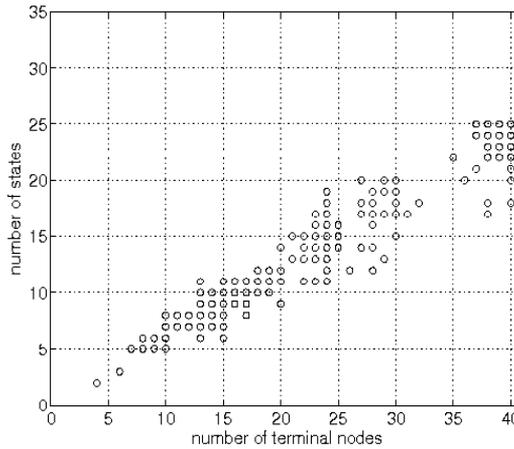


Figure 12. Relation between terminal nodes and internal states (genetic programming controllers are evolved with a varying number of terminal nodes)

For the next experiment, we set up the number of states as a control parameter. In a similar process as above, we test a varying number of states. If an evolved tree includes more states than the limit, the tree will be mutated until the limit condition is satisfied. In addition, we set the maximum number of terminal nodes for evolving trees to $1.7 n_s$ for each number of states, n_s , using the above relation rule between terminal nodes and states (Fig. 12). Without the restriction on terminal nodes, the evolving trees encountered bloat, degrading the performance. As shown in Fig. 11(b), the internal states play a critical role on the performance improvement, and the fitness performance is enhanced with more internal states. However, for a given number of states, the genetic programming result is mostly worse than the FSM evolution result shown in Fig. 5. The best fitness that the genetic programming achieved for the overall experiments was 44, and it had a similar performance

with the FSM evolution only for a few sets of internal states. Evolving directly the FSM structure tends to produce more efficient controllers and its performance level is significantly better. The S-expression is a procedural program arranging a sequence of actions, while FSMs can not only encode a sequence of actions, but also they have more dynamic features in representation by allowing flexible transitions and actions from state to state.

By the fitness distribution of genetic programming controllers, we built a partial order relation among memory structure as shown in Figure 13. The Wilcoxon rank-sum test and beta distribution test produce almost the same diagram of partial order relations except for the case between 10 states and 12 states. One reason for this is that the Wilcoxon test measures the rank sum over all the observations which belong to one sample group, whilst the beta distribution focuses on what level of good fitness performance one sample group reach. When the diagram is compared with the partial order graph by the FSM result (Fig. 9), the threshold level for the relation is changed, but roughly keeps the relation structure. It seems that a large number of states have more variation on the relation result. As a matter of fact, the lattice diagram for memory analysis is extracted from empirical data which depends on the corresponding control structure. It only shows the partial order information among memory structure with a given confidence level, but it does not mean that the corresponding structure guarantees a given level of performance or it cannot achieve better performance than the threshold level. More trial runs will support more reliable information of the performance level or the partial order relation.

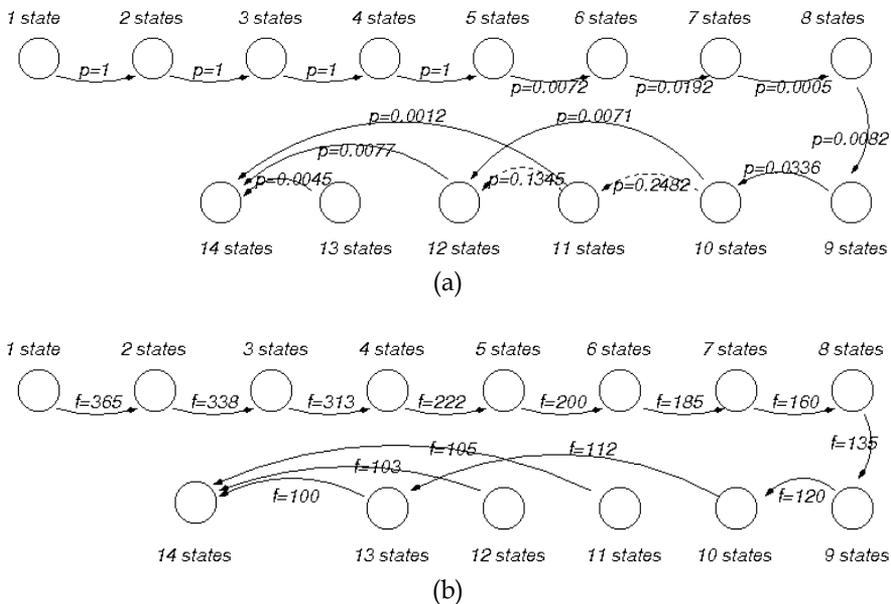


Figure 13. A partial order relation of S-expression trees with varying number of states in terms of behaviour performance (a) Wilcoxon rank-sum test (solid: significant difference, dashed: insignificant difference) (b) beta distribution test (the arrow label indicates the threshold fitness for success/failure)

So far we showed the distribution of success rate or computational effort to measure the performance difference between a pair of sample groups. The significance test result of the success rate is equivalent to that of the effort cost if almost the same computing cost is spent for each evolutionary run. At this point the analysis of the effort cost is not more helpful than the analysis of success rate. However, the information of computational effort can be used to understand the evolutionary process for a given problem. Table 2 shows the number of successful runs among 50 trials and the computational effort for each configuration.

To reach the fitness level 150, FSM controllers obtained 17 successes with 6 internal states and 10^5 evaluations (2,000 generations). More generations, for example, 5,000 generations and 10,000 generations produced more successes as expected. The unit cost C_2 , C_5 , C_{10} is the computing cost for 10^5 , 2.5×10^5 , 5×10^5 evaluations and the costs can be approximated with $C_{10} = 2C_5 = 5C_2$. Then the confidence intervals for 10^5 , 2.5×10^5 , 5×10^5 (2,000, 5,000 and 10,000 generations) become $[1.09C_2, 3.46C_2]$, $[1.69C_2, 5.09C_2]$, $[1.74C_2, 5.40C_2]$, respectively. Then the experiments with 2,000 generations have the confidence interval with the smallest effort level, whose range is also narrow. Thus, we can recommend 2,000 generations for the future experiments. For 11 states with a desirable fitness of 50, 10^5 , 2.5×10^5 , 5×10^5 evaluations produce the confidence intervals $[5.16C_2, 44.91C_2]$, $[2.51C_2, 7.86C_2]$ and $[3.12C_2, 9.39C_2]$, respectively. For this case, 2.5×10^5 evaluations (5,000 generations) would be a better choice for evolution.

Parameters			genome evaluations					
pop.	states	fitness	$\mu \leq 10^5$		$\mu \leq 2.5 \times 10^5$		$\mu \leq 5 \times 10^5$	
100	5	222	48	$[0.01C_2, 0.16C_2]$	48	$[0.01C_5, 0.16C_5]$	48	$[0.01C_{10}, 0.16C_{10}]$
100	6	150	17	$[1.09C_2, 3.46C_2]$	23	$[0.67C_5, 2.03C_5]$	31	$[0.35C_{10}, 1.08C_{10}]$
100	7	120	22	$[0.73C_2, 2.21C_2]$	33	$[0.29C_5, 0.92C_5]$	37	$[0.19C_{10}, 0.65C_{10}]$
100	8	150	45	$[0.05C_2, 0.27C_2]$	46	$[0.03C_5, 0.23C_5]$	47	$[0.02C_{10}, 0.19C_{10}]$
100	8	100	21	$[0.79C_2, 2.41C_2]$	32	$[0.32C_5, 1.00C_5]$	41	$[0.11C_{10}, 0.45C_{10}]$
100	9	70	7	$[2.81C_2, 13.24C_2]$	17	$[1.09C_5, 3.46C_5]$	22	$[0.73C_{10}, 2.21C_{10}]$
100	10	50	2	$[6.43C_2, 80.30C_2]$	10	$[2.02C_5, 7.86C_5]$	19	$[0.93C_{10}, 2.87C_{10}]$
100	11	50	3	$[5.16C_2, 44.91C_2]$	18	$[1.00C_5, 3.14C_5]$	24	$[0.62C_{10}, 1.88C_{10}]$
100	15	25	1	$[8.57C_2, 208.21C_2]$	6	$[3.19C_5, 16.54C_5]$	19	$[0.93C_{10}, 2.87C_{10}]$

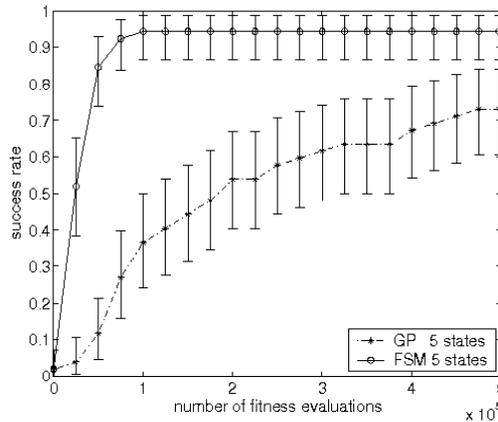
(a)

Parameters			genome evaluations					
pop.	states	fitness	$\mu \leq 10^7$		$\mu \leq 2.5 \times 10^7$		$\mu \leq 5 \times 10^7$	
500	5	222	18	$[1.00K_{0.4}, 3.14K_{0.4}]$	29	$[0.42K_1, 1.26K_1]$	37	$[0.19K_2, 0.66K_2]$
500	6	200	15	$[1.28K_{0.4}, 4.23K_{0.4}]$	23	$[0.68K_1, 2.04K_1]$	28	$[0.45K_2, 1.36K_2]$
500	7	180	5	$[3.67K_{0.4}, 21.51K_{0.4}]$	12	$[1.67K_1, 5.98K_1]$	13	$[1.52K_2, 5.29K_2]$
500	8	150	5	$[3.67K_{0.4}, 21.51K_{0.4}]$	6	$[3.19K_1, 16.54K_1]$	9	$[2.23K_2, 9.18K_2]$
500	9	150	7	$[2.81K_{0.4}, 13.24K_{0.4}]$	12	$[1.67K_1, 5.98K_1]$	17	$[1.09K_2, 3.46K_2]$
500	10	120	5	$[3.67K_{0.4}, 21.51K_{0.4}]$	10	$[2.02K_1, 7.86K_1]$	11	$[1.83K_2, 6.82K_2]$
500	11	120	7	$[2.81K_{0.4}, 13.24K_{0.4}]$	13	$[1.52K_1, 5.29K_1]$	15	$[1.28K_2, 4.23K_2]$

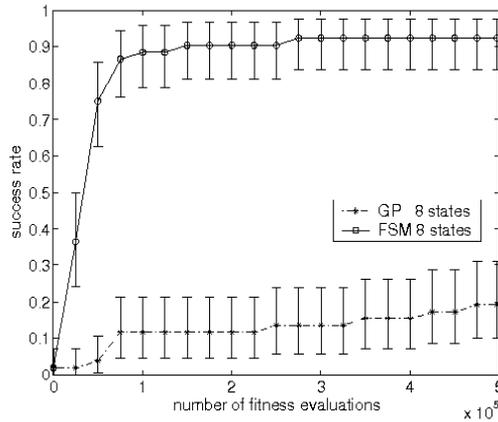
(b)

Table 2. Effort costs for exploration time with number of genome evaluations (a) evolving FSM controllers (b) evolving S-expression trees; C_i , K_i is the basic unit effort cost of a single run which fails to reach the corresponding fitness (C_i , K_i slightly varies on different fitness levels, but for simplicity the same notation is used). Each pair of values represent the number of successes among 50 experiments and its 95% confidence interval for the effort cost

We can compare the computational efforts for different types of controllers or different algorithms. The analysis of the effort cost will be useful especially when the algorithms have different CPU run-time for an evolutionary run. For instance, the FSM controller with eight states and 10^5 evaluations (2,000 generations) has a confidence interval $[0.05C_2, 0.27C_2]$ to obtain the fitness 150, while the genetic programming for eight states needs a computing cost $[3.67K_{0.4}, 21.51 K_{0.4}]$ to obtain the first success with 95% chance. If we know the computing costs for a single run, C_2 and $K_{0.4}$ we can take the significance test over the two strategies by comparing the confidence intervals. Indeed, the ratio of the average CPU run-time between the FSM and the genetic programming ($K_{0.4} / C_2$) was 1.23 in the experiments. The comparison result implies that the FSM controllers produce more efficient results than the genetic programming controllers. This is confirmed again in other pairwise tests for small fitness in Table 2.



(a)



(b)

Figure 14. Run-time distribution with genetic programming and FSM controllers (the average and 95% confidence range of success rate in the beta distribution are displayed) (a) evolving five internal states (threshold fitness: 222) (b) evolving eight internal states (threshold fitness: 150)}

Hoos and Stuetzle (1998) studied a run-time distribution to compare different algorithms or determine parameter settings. The distribution shows the empirical success rate depending on varying run time. In our experiments, the number of trial runs is relatively small and so the beta distribution of success rate was applied to the run-time distribution as shown in Fig. 14, where the number of fitness evaluations was used instead of the actual CPU run-time. The FSMs with five and eight states showed significantly better performance in the run-time process. Generally, FSMs show more discriminative performance for a large number of states.

5. Discussion

The number of states in finite state machines in the experiments may not be exactly the same as the number of states that the evolved controllers actually use for exploration. It specifies a maximum limit over the number of finite states. Especially for a large number of states, controllers that do not use all memory states are sometime evolved even though a given maximum memory limit is specified. The same can be true of the genetic programming structure. When the maximum number of terminal nodes was set up for evolutionary runs, some best controllers used a smaller number of nodes than the limit size, or had a redundant expression. Thus, our analysis of memory states may have a little discrepancy with the actual usage of memory.

Genetic programming has a high-level representation feature with a procedural program. When an S-expression is translated into a finite automaton, it has a main loop for repeating the action sequence. It often has a sequential process among internal states until the end of program is reached. In contrast, the Mealy machine notation allows transition loops among internal states. Evolving the FSM controllers can create such loops for in-between states (from state to state) and more conditional transition branches. The flexible representation of the Mealy machine provides more dynamic property for a given number of states. The performance difference between the two types of controllers is due to the characteristics of representation.

To discriminate the performances of a varying number of internal states, the beta distribution of success rate or computational effort was used. We believe that the success rate is a better criterion for this application, because we are more interested in the on-off decision of the quality of controllers with a given evolutionary setting rather than efficient development of controllers. The computational effort can be more effective when strategies to be compared have different computing costs or when the efficiency is a major criterion in the evolutionary experiments. An assumption for the suggested significance test of computational effort is that each single run has almost the same level of computing cost. If each run may have a significantly different computing cost, the estimated computational effort based on success rate would have a deviation from the actual effort. The run-time distribution, that is, the curve of success rate for variable computing cost provides the characteristics of a given algorithm and we can easily observe the transition of performance with run-time. The run-time distribution with its confidence range would be a useful tool to compare different algorithms.

In the evolutionary computation research, the performance comparison among evolutionary algorithms has often used the average performance over fitness samples or *t*-statistic. We argue that the comparison without observing the fitness distribution may not notice significant difference. The beta distribution analysis or Wilcoxon rank-sum test would

provide more precise information than *t*-distribution when the number of samples is small or the fitness samples are influenced by the deceptive property of a given problem. We applied the Wilcoxon rank-sum test and the beta distribution for partial order relations between a varying number of internal states with FSMs. The beta distribution test with 50 trials produced the same significance result as the well-known Wilcoxon test and it indirectly supports the validity of the suggested beta distribution. An advantage of the beta distribution is that it can estimate the computational effort and success rate without difficulty, as well as do significance test over a pair of sample groups. It attends to how many observations for a desirable level of performance are found consistently with a given strategy rather than considers the whole distribution of observations. If an evolutionary search problem is deceptive, a skewed or bimodal distribution of fitness samples can be observed. The above results imply that the non-parametric tests over the distribution of fitness samples would be more effective.

The memory analysis and methods suggested in this paper can be applied to agent behaviours in a grid world. The tested ant problem has a restricted set of sensory configurations and motor actions. However, the agents in the real world, which often includes a variety of sensory apparatus and motor actions, have different characteristics or behaviours. Noisy sensor readings may need more dynamic configurations or structures. The FSM control structure has a representation problem for a large number of input conditions and a large scale of motor actions, although the amount of internal memory is easily quantifiable. Thus, the current approach may have a limitation in real-world problems. In the experiments we tested a single environment, but it may need more variety of environmental configurations to design robust controllers. Then the memory condition for the ant problem will be changed. We leave this work for future study.

6. Conclusion

The artificial ant problem is an agent task to model ant trail following in a grid world, and the environment has a perceptual aliasing problem. The ant problem is a good example to see memory effects in behavior performance. An artificial ant agent must seek all the food lying in a grid and follow the irregular trail for the food. Relatively many memory states are required to remember environmental features. Its performance is measured by how many time steps are required for an ant agent to collect all food on a trail.

We applied two types of memory encoding controllers, finite state machines and genetic programming controllers for the analysis of internal memory, where the genetic programming controllers are transformable into FSMs. We first explored a systematic analysis over the usage of internal memory, based on statistical significance test and tried to identify the role of internal states. The internal states needed for a given threshold performance are easily quantifiable with FSMs. We obtained a partial order of memory states necessary with a variety of performance levels. The results show that the ant problem needs at least five internal states. To develop more efficient strategies (collecting all food in a shorter time) in the artificial ant problem, agents need more memory states according to the memory analysis and also there is a limit level of memory over which the performance is rarely improved. Using the suggested approach, we can identify the role of internal states or observe the relevance of memory to agent behaviours. This will help us understand the characteristics of agent behaviours by decomposing the behaviours in terms of internal states.

Three different statistic tests, t -test, Wilcoxon rank-sum, and beta distribution, were applied to discriminate the performance difference among a varying number of internal states. The beta distribution test has a good precision of significance test, and its test result is similar to that of the Wilcoxon test. In many cases, the beta distribution test of success rate was useful where the t -test could not discriminate the performance. The beta distribution test based on sampling theory has an advantage on analyzing the fitness distribution with even a small number of evolutionary runs, and it has much potential for application as well as provide the computational effort. In addition, the method can be applied to test the performance difference of an arbitrary pair of methodologies. The estimation of computational effort provides the information of an expected computing time for success, or how many trials are required to obtain a solution. It can also be used to evaluate the efficiency of evolutionary algorithms with different computing time.

We compared genetic programming approach and finite state machines, and the significance test with success rate or computational effort shows that FSMs have more powerful representation to encode internal memory and produce more efficient controllers than the tree structure, while the genetic programming code is easy to understand.

7. References

- P.J. Angeline, G.M. Saunders, and J.B. Pollack (1994). An evolutionary algorithm that constructs recurrent neural networks, *IEEE Trans. on Neural Networks*, 5(1): pp. 54-65.
- P.J. Angeline (1998). Multiple interacting programs: A representation for evolving complex Behaviors, *Cybernetics and Systems*, 29(8): pp. 779-806.
- D. Ashlock (1997). GP-automata for dividing the dollar, *Genetic Programming 97*, pp. 18-26. MIT Press
- D. Ashlock (1998). ISAc lists, a different representation for program induction, *Genetic Programming 98*, pp. 3-10. Morgan Kaufman.
- B. Bakker and M. de Jong (2000). The epsilon state count. *From Animals to Animats 6: Proceedings of the Sixth Int. Conf. on Simulation of Adaptive Behaviour*, pp. 51-60. MIT Press
- K. Balakrishnan and V. Honavar (1996). On sensor evolution in robotics. *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 455--460, Stanford University, CA, USA. MIT Press.
- D. Braziunas and C. Boutilier (2004). Stochastic local search for POMDP controllers. *Proc. of AAAI*, pages 690--696
- S. Christensen and F. Oppacher (2002). An analysis of Koza's computational effort statistics, *Proceedings of European Conference on Genetic Programming*, pages 182-191.
- P. R. Cohen (1995), *Empirical methods for artificial intelligence*, MIT Press, Cambridge, Mass., 1995.
- M. Colombetti and M. Dorigo (1994), Training agents to perform sequential behavior, *Adaptive Behavior*, 2 (3): 305-312.
- J. Elman (1990). Finding structure in time. *Cognitive Science*, 14: 179-211.
- L.J. Fogel, A.J. Owens, and M.J. Walsh (1996), *Artificial intelligence through simulated evolution*, Wiley, New York, 1966.

- H.H. Hoos and T. Stuetzle (1998). Evaluating LasVegas algorithms - pitfalls and remedies, *Proceedings of the 14th Conf. on Uncertainty in Artificial Intelligence*, pages 238--245. Morgan Kaufmann
- H.H. Hoos and T. Stuetzle (1999). Characterising the behaviour of stochastic local search, *Artificial Intelligence*, 112 (1-2): 213--232.
- J.E. Hopcroft and J.D. Ullman (1979). *Introduction to automata theory, languages, and computation*. Addison Wesley, Reading, MA.
- D. Jefferson, R. Collins, C. Cooper, M. Dyer, M. Flowers, R. Korf, C. Taylor, and A. Wang (1991). Evolution as a theme in artificial life, *Artificial Life II*. Addison Wesley.
- D. Kim and J. Hallam (2001). Mobile robot control based on Boolean logic with internal Memory, *Advances in Artificial Life, Lecture Notes in Computer Science* vol. 2159, pp. 529-538.
- D. Kim and J. Hallam (2002). An evolutionary approach to quantify internal states needed for the Woods problem, From Animals to Animats 7, *Proceedings of the Int. Conf. on the Simulation of Adaptive Behavior*, pages 312-322. MIT Press
- D. Kim (2004). Analyzing sensor states and internal states in the tartarus problem with tree state machines, *Parallel Problem Solving From Nature 8, Lecture Notes on Computer Science* vol. 3242, pages 551-560.
- D. Kim (2006). Memory analysis and significance test for agent behaviours, *Proc. of Genetic and Evolutionary Computation Conf. (GECCO)*, pp. 151-158.
- Z. Kohavi (1970). *Switching and Finite Automata Theory*, McGraw-Hill, New York, London.
- J. R. Koza (1992). *Genetic Programming*, MIT Press, Cambridge, MA.
- W.B. Langdon and R. Poli (1998). Why ants are hard, *Proceedings of Genetic Programming*.
- P.L. Lanzi.(1998). An analysis of the memory mechanism of XCSM, *Genetic Programming* 98, pages 643--651. Morgan Kauffman
- P.L. Lanzi (2000). Adaptive agents with reinforcement learning and internal memory, *From Animals to Animats 6: Proceedings of the Sixth Int. Conf. on Simulation of Adaptive Behaviour*, pages 333-342. MIT Press.
- W.-P. Lee (1998) *Applying Genetic Programming to Evolve Behavior Primitives and Arbitrators for Mobile Robots*, Ph. D. dissertation, University of Edinburgh.
- L. Lin and T. M. Mitchell (1992). Reinforcement learning with hidden states, *From Animals to Animats 2: Proceedings of the Second Int. Conf. on Simulation of Adaptive Behaviour*, pages 271--280. MIT Press.
- A.K. McCallum (1996). *Reinforcement Learning with Selective Perception and Hidden State*, Ph.D. dissertation, University of Rochester.
- N. Meuleau, L. Peshkin, K.-E. Kim, and L. P. Kaelbling (1999). Learning finite-state controllers for partially observable environments. *Proc. of the Conf. on UAI*, pages 427--436.
- J.H. Miller. The coevolution of automata in the repeated prisoner's dilemma, *Journal of Economics Behavior and Organization*, 29(1): 87-112.
- Jr. R. Miller (1986). *Beyond ANOVA, Basics of Applied Statistics*, John Wiley & Sons, New York.
- J. Niehaus and W. Banzhaf (2003). More on computational effort statistics for genetic programming. *Proceedings of European Conference on Genetic Programming*, pages 164-172.
- S. Nolfi and D. Floreano (2000). *Evolutionary Robotics : The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, Cambridge, MA.
- L. Peshkin, N. Meuleau, L.P. Kaelbling (1999). Learning policies with external memory, *Proc. of Int. Conf. on Machine Learning*, pp. 307-314, 1999

-
- S.M. Ross (2000). *Introduction to Probability and Statistics for Engineers and Scientists*. Academic Press, San Diego, CA, 2nd edition.
- A. Silva, A. Neves, and E. Costa (1999). Genetically programming networks to evolve memory mechanism, *Proceedings of Genetic and Evolutionary Computation Conference*.
- E.A. Stanley, D. Ashlock, and M.D. Smucker (1995). Iterated prisoner's dilemma game with choice and refusal of partners, *Advances in Artificial Life : Proceedings of European Conference on Artificial Life*.
- A. Teller (1994). The evolution of mental models, *Advances in Genetic Programming*, MIT Press.
- C. Wild and G. Seber (1999). *Chance Encounters: A First Course in Data Analysis and Inference*. John Wiley & Sons, New York.
- S.W. Wilson (1994). ZCS: A zeroth level classifier system, *Evolutionary Computation*, 2 (1): 1-18.

Evolutionary Parametric Identification of Dynamic Systems

Dimitris Koulocheris and Vasilis Dertimanis
*National Technical University of Athens
Greece*

1. Introduction

Parametric system identification of dynamic systems is the process of building mathematical, time domain models of plants, based on excitation and response signals. In contrast to its nonparametric counterpart, this model based procedure leads to fixed descriptions, by means of finitely parameterized transfer function representations. This fact provides increased flexibility and makes model-based identification a powerful tool with growing significance, suitable for analysis, fault diagnosis and control applications (Mrad et al, 1996, Petsounis & Fassois, 2001).

Parametric identification techniques rely mostly on Prediction-Error Methods (Ljung, 1999). These methods refer to the estimation of a certain model's parameters, through the formulation of one-step ahead prediction errors sequence, between the actual response and the one computed from the model. The evaluation of prediction errors is taking place throughout the mapping of the sequence to a scalar-valued index function (loss function). Over a set of candidate sets with different parameters, the one which minimizes the loss function is chosen, with respect to the corresponding fitness to data. However, in most cases the loss function cannot be minimized analytically, due to the non-linear relationship between the parameter vector and the prediction-error sequence. The solution then has to be found by iterative, numerical techniques. Thus, PEM turns into a non-convex optimization problem, whose objective function presents many local minima.

The above problem has been mostly treated so far by deterministic optimization methods, such as Gauss-Newton or Levenberg-Marquardt algorithms. The main concept of these techniques is a gradient-based, local search procedure, which requires smooth search space, good initial "guess", as well as well-defined derivatives. However, in many practical identification problems, these requirements often cannot be fulfilled. As a result, PEM stagnate to local minima and lead to poorly identified systems.

To overcome this difficulty, an alternative approach, based in the implementation of stochastic optimization algorithms, has been developed in the past decade. Several techniques have been formulated for parameter estimation and model order selection, using mostly Genetic Algorithms. The basic concept of these algorithms is the simulation of a natural evolution for the task of global optimization, and they have received considerable interest since the work done (Kristinsson & Dumont, 1992), who applied them to the identification of both continuous and discrete time systems. Similar studies are reported in literature (Tan & Li, 2002, Gray et al. , 1998, Billings & Mao, 1998, Rodriguez et al., 1997). Fleming & Purshouse, 2002 have presented an extended survey on these techniques, while Schoenauer & Sebag, 2002 address the use of domain knowledge and the choice of fitting

functions in Evolutionary System Identification. Yet, most of these studies are limited in scope, as they, almost exclusively, use Genetic Algorithms or Genetic Programming for the various identification tasks, they mostly refer to non-linear model structures, while test cases of dynamic systems are scarcely used. Furthermore, the fully stochastic nature of these algorithms frequently turns out to be computationally expensive, since they cannot assure convergence in a standard number of iterations, thus leading to extra uncertainty in the quality of the estimation results.

This study aims at interconnecting the advantages of deterministic and stochastic optimization methods in order to achieve globally superior performance in PEM. Specifically, a hybrid optimization algorithm is implemented in the PEM framework and a novel methodology is presented for the parameter estimation problem. The proposed method overcomes many difficulties of the above mentioned algorithms, like stability and computational complexity, while no initial "guess" for the parameter vector is required. For the practical evaluation of the new method's performance, a testing apparatus has been used, which consists of a flexible robotic arm, driven by a servomotor, and a corresponding data set has been acquired for the estimation of a Single Input-Single Output (SISO) ARMAX model. The rest of the paper is organized as follows: In Sec. 2 parametric system identification fundamentals are introduced, the ARMAX model is presented and PEM is been formatted in it's general form. In Sec. 3 optimization algorithms are discussed, and the hybrid algorithm is presented and compared. Section 4 describes the proposed method for the estimation of ARMAX models, while in Sec. 5 the implementation of the method to parametric identification of a flexible robotic arm is taking place. Finally, in Sec. 6 the results are discussed and concluding remarks are given.

2. Parametric identification fundamentals

Consider a linear, time-invariant and casual dynamic system, with a single input and a single output, described by the following equation in the z-domain (Oppenheim & Schaffer, 1989),

$$Y(z) = H(z) \cdot X(z) \quad (1)$$

where $X(z)$ and $Y(z)$ denote the z-transforms of input and output respectively, and $H(z)$ is a rational transfer function, with respect to the variable z , which describes the input-output dynamics. It should be noted that the selection of representing the true system in the z-domain is justified from the fact that data are always acquired in discrete time units. Due to one-to-one relationship between the z-transform and it's Laplace counterpart, it is easy to obtain a corresponding description in continuous time.

The identification problem pertains to the estimation of a finitely parameterized transfer function model of a given structure, similar to that of $H(z)$, by means of the available data set and taking under consideration the presence of noisy measurements. The estimated model must have similar properties to that of the true one, it should be able to simulate the dynamic system and, additionally, to predict future values of the output. Among a large number of ready-made models (known also as black-box models), ARMAX is widespread and has performed well in many engineering applications (Petsounis & Fassois, 2001).

2.1 The ARMAX model structure

A SISO ARMAX(na, nb, nc, nk) model has the following mathematical representation

$$A(q) \cdot y_t = B(q) \cdot u_t + C(q) \cdot e_t \quad (2)$$

where u_t and y_t represent the sampled excitation and noise corrupted response signals, for time $t = 1, \dots, N$ respectively and e_t is a white noise sequence with $E\{e_t\} = 0$ and $E\{e_t \cdot e_{t+\tau}\} = \delta_\tau \cdot \sigma_e^2$, where δ_τ and σ_e^2 are Kronecker's delta and white noise variance respectively. N is the number of available data, q denotes the backshift operator, so that $y_t \cdot q^k = y_{t-k}$, and $A(q)$, $B(q)$, $C(q)$ are polynomials with respect to q , having the following form

$$A(q) = 1 + a_1 \cdot q^{-1} + \dots + a_{na} \cdot q^{-na} \quad (3)$$

$$B(q) = (b_1 + b_2 \cdot q^{-1} + \dots + b_{nb} \cdot q^{-nb+1}) \cdot q^{-nk} \quad (4)$$

$$C(q) = 1 + c_1 \cdot q^{-1} + \dots + c_{nc} \cdot q^{-nc} \quad (5)$$

The term q^{-nk} in (4) is optional and represents the delay from input to output.

In literature, the full notation for this specific model is ARMAX(na, nb, nc, nk), and it is totally described by the order of the polynomials mentioned above, the numerical values of their coefficients, the delay nk , as well as the white noise variance σ_e^2 .

In Eq. (2) it is obvious that ARMAX consists of two transfer functions, one between input and output

$$G(q) = \frac{B(q)}{A(q)} \quad (6)$$

which models the dynamics of the system, and one between noise and output

$$H(q) = \frac{C(q)}{A(q)} \quad (7)$$

which models the presence of noise in the output. For a successful representation of a dynamic system, by means of ARMAX models, the stability of the above two transfer functions is required. This can be achieved by letting the roots of $A(q)$ polynomial lie outside the unit circle with zero origin, in the complex plane (Ljung, 1999, Oppenheim & Schaffer, 1989). In fact, there is an additional condition that must hold and that is the invertibility of the noise transfer function $H(q)$ (Ljung, 1999, Box et al., 1994, Soderstrom & Stoica, 1989). For this reason, $C(q)$ polynomial must satisfy the same requirement as $A(q)$.

2.2 Formulation of PEM

For a given data set over the time t , it is possible to compute the output y_t of an ARMAX model, at time $t + 1$. This fact yields, for every time instant, to the formulation of one step ahead prediction-errors sequence, between the actual system's response and the one computed by the model

$$\hat{e}_t(1/p) = y_{t+1} - \hat{y}_t(1/p) \quad (8)$$

where $p=[a_i \ b_i \ c_i]$ is the parameter vector to be estimated, for given orders n_a , n_b , n_c and delay n_k , y_{t+1} the measured output, $\hat{y}_t(1/p)$ the model's output and $\hat{\varepsilon}_t(1/p)$ the prediction error (also called model residual). The argument $(1/p)$ denotes conditional probability (Box et al., 1994) and the hat indicates estimator/estimate.

The evaluation of residuals is implemented through a scalar-valued function (see Introduction), which in general has the following form

$$V_N = \frac{1}{N} \cdot \sum_{t=0}^N \hat{\varepsilon}_t(1/p) \quad (9)$$

Obviously, the parameter p which minimizes V_N is selected as the most suitable

$$p = \operatorname{argmin}\{V_N\} \quad (10)$$

Unfortunately, V_N cannot be minimized analytically due to the non-linear relationship between the model residuals $\hat{\varepsilon}_t(1/p)$ and the parameter vector p . This can be noted, by writing (2) in a slightly different form

$$\hat{\varepsilon}_t(1/p) = \frac{A(q)}{C(q)} \cdot \mathcal{Y}_t - \frac{B(q)}{C(q)} \cdot u_t \quad (11)$$

The solution then has to be found by iterative, numerical techniques and this is the reason for the implementation of optimization algorithms within the PEM framework.

3. Optimization algorithms

In this section, the hybrid optimization algorithm is presented. The new method is a combination of a stochastic and a deterministic algorithm. The stochastic component belongs to the Evolutionary Algorithms (EA's) and the deterministic one to the quasi-Newton methods for optimization.

3.1 Evolution strategies

In general, EA's are methods that simulate natural evolution for the task of global optimization (Baeck, 1996). They originate in the theory of biological evolution described by Charles Darwin. In the last forty years, research has developed EA's so that nowadays they can be clearly formulated with very specific terms. Under the generic term Evolutionary Algorithms lay three categories of optimization methods. These methods are Evolution Strategies (ES), Evolutionary Programming (EP) and Genetic Algorithms (GA) and share many common features but also approximate natural evolution from different points of view.

The main features of ES are the use of floating-point representation for the population and the involvement of both recombination and mutation operators in the search procedure. Additionally, a very important aspect is the deterministic nature of the selection operator. The more advanced and powerful variations are the multi-membered versions, the so-called $(\mu+\lambda)$ -ES and (μ,λ) -ES which present self-adaptation of the strategy parameters.

3.2 The quasi-Newton BFGS optimization method

Among the numerous deterministic optimization techniques, quasi-Newton methods are combining accuracy and reliability in a high level (Nocedal & Wright, 1999). They are derived from the Newton's method, which uses a quadratic approximation model of the objective function, but they require significantly less computations of the objective function during each iteration step, since they use special formulas in order to compute the Hessian matrix. The decrease of the convergence rate is negligible. The most popular quasi-Newton method is the BFGS method. This name is based on its discoverers Broyden, Fletcher, Goldfarb and Shanno (Fletcher, 1987).

3.3 Description of the hybrid algorithm

The optimization procedure presented in this paper focuses in interconnecting the advantages presented by EA's and mathematical programming techniques, and aims at combining high convergence rate with increased reliability in the search for the global optimum in real parameter optimization problems. The proposed algorithm is based on the distribution of the local and the global search for the optimum. The method consists of a super-positioned stochastic global search and an independent deterministic procedure, which is activated under conditions in specific members of the involved population. Thus, while every member of the population contributes in the global search, the local search is realized from single individuals. Similar algorithmic structures have been presented in several fully stochastic techniques that simulate biological procedures of insect societies. Such societies are distributed systems that, in spite of the simplicity of their individuals, present a highly structured social organization. As a result, such systems can accomplish complex tasks that in most cases far exceed the individual's capabilities. The corresponding algorithms use a population of individuals, which search for the optimum with simple means. The synthesis, though, of the distributed information enables the overall procedure to solve difficult optimization problems. Such algorithms were initially designed to solve combinatorial problems (Dorigo et al., 2000), but were soon extended to optimization problems with continuous parameters (Monamarche et al., 2000, Rjesh et al., 2001). A similar optimization technique presenting a hybrid structure has been already discussed in (Kananachos, 2002), and it's based on a mechanism that realizes cooperation between the (1,1)-ES and the Steepest Descent method.

The proposed methodology is based on a mechanism that aims at the cooperation between the $(\mu+\lambda)$ -ES and the BFGS method. The conventional ES (Baeck, 1996, Schwefel, 1995), is based on three operators that take on the recombination, mutation and selection tasks. In order to maintain an adequate stochastic character of the new algorithm, the recombination and selection operators are retained with out alterations. The improvement is based on the substitution of the stochastic mutation operator by the BFGS method. The new deterministic mutation operator acts only on the ν non-privileged individuals in order to prevent loss of information from the corresponding search space regions, while three other alternatives were tested. In these, the deterministic mutation operator is activated by:

- every individual of the involved population,
- a number of privileged individuals, and
- a number of randomly selected individuals.

The above alternatives led to three types of problematic behavior. Specifically, the first alternative increased the computational cost of the algorithm without the desirable effect.

The second alternative led to premature convergence of the algorithm to local optima of the objective function, while the third generated unstable behavior that led to statistically low performance.

3.4 Efficiency of the hybrid algorithm

The efficiency of the hybrid algorithm is compared to that of the (15 +100)-ES, the (30, 0.001, 5, 100)GA, as well as the (60, 10, 100)meta-EP method, for the Fletcher & Powell test function, with twenty parameters. Progress of all algorithms is measured by base ten logarithm of the final objective function value

$$P = \log_{10}(f_{final}) \quad (12)$$

Figure 1 presents the topology of the Fletcher & Powell test function for $n = 2$.

The maximum number of objective function evaluations is $2 \cdot 10^5$. In order to obtain statistically significant data, a sufficiently large number of independent tests must be performed. Thus, the results of $N = 100$ runs for each algorithm were collected. The expectation is estimated by the average:

$$\bar{P} = \frac{1}{N} \cdot \sum_{i=1}^N P_i \quad (13)$$

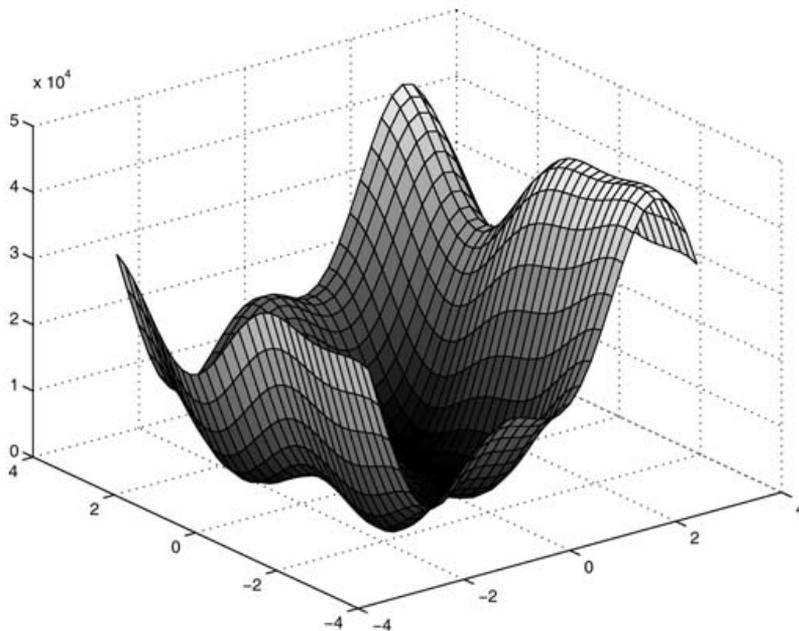


Figure 1. The Fletcher & Powell test function

The results are presented in Table 1.

Test Results	\bar{P}	$\min_{1 \leq i \leq 100} P_i$	$\max_{1 \leq i \leq 100} P_i$
Hybrid	-7.15	-8.98	3.12
ES	3.94	2.07	5.20
EP	4.13	3.14	5.60
GA	4.07	3.23	5.05

Table 1. Results on the Fletcher & Powell function for $n = 20$

4. Description of the proposed method

The proposed method for the parameter estimation of ARMAX(na, nb, nc, nk) models consists of two stages. In the first stage, Linear Least Squares are used to estimate an ARX(na, nb, nk) model of the form

$$A(q) \cdot y_t = B(q) \cdot u_t + \varepsilon_t \quad (14)$$

based upon the observation that the nonlinear relationship between the model residuals and the parameter vector would be overcome if $C(q)$ polynomial was monic (see (11)). Considering the same loss function, as in (9), by expressing the ARX model in (14) as

$$y_t = \varphi_t^T \cdot p^* + \varepsilon_t \quad (15)$$

with

$$\varphi_t^T = [y_{t-1} \dots y_{t-na} \ u_t \dots u_{t-nb}] \quad (16)$$

being the regression vector and $p^* = [a_i \ b_i]$ the parameter vector, the minimizing argument for the model in (14) can be found analytically, by setting the gradient of V_N equal to zero, which yields

$$\hat{p}_{\min}^* = \left[\frac{1}{N} \cdot \sum_{t=1}^N \varphi_t \cdot \varphi_t^T \right]^{-1} \cdot \left[\frac{1}{N} \cdot \sum_{t=1}^N \varphi_t \cdot y_t \right] \quad (17)$$

The parameter vector p , as computed from (17), can be used as a good starting point for the values of $A(q)$ and $B(q)$ coefficients. In other estimation methods, like the two Stage Least Squares or the Multi-Stage Least Squares (Petsounis & Fassois, 2001) the ARX model is estimated with sufficiently high orders. In the presented method this is not necessary, since the resulted from (17) values are constantly optimized within the hybrid algorithm. Additionally, this stage cannot be viewed as initial "guess", since the information that is used does not deal with the ARMAX model in question.

It is rather an adaptation of the hybrid optimization algorithm to become problem specific. In the second stage the ARMAX(na, nb, nc, nk) model is estimated by means of the hybrid algorithm. The parameter vector now becomes

$$\hat{\vartheta} = [\hat{p}_{\min}^* \ c_i] \quad (18)$$

with c_i denoting the additional parameters, due to the presence of $C(q)$ polynomial. The values c_i are randomly chosen from the normal distribution. The hybrid algorithm is presented below,

```

counter j :=0;
initialize  $\hat{p}(0)$ 
evaluate  $\hat{p}(0)$ 
while T{ $\hat{p}(j)$ } ≠ true do
  recombine:  $\hat{p}'(j) = r(\hat{p}(j))$ 
  evaluate:  $\hat{p}'(j)$ 
  mutate:  $\hat{p}''(j) = m(\hat{p}'(j))$ 
  evaluate:  $\hat{p}''(j)$ 
  select:  $\hat{p}(j+1) = s(\hat{p}''(j))$ 
  j:=j+1
end while

```

where j is the iteration counter, $\hat{p}(j)$ the current population, T the termination criterion, r the recombination operator, m the mutation operator (provided by the BFGS) and s the selection operator (see Sec. 3). The evaluation of parameter vector at each iteration is realized via the calculation of objective function.

For the successful realization of the hybrid algorithm, two issues must be further examined: the choice of the predictor, which modulates the residual sequence and the choice of the objective function, from which this sequence is evaluated at each iteration. An additional topic is the choice of the “best” model among a number of estimated ones. This topic is covered by statistical tests for order selection.

4.1 Choice of predictor

It is obvious that in every iteration of the hybrid algorithm the parameter vector $\hat{p}(j)$ is evaluated, in order to examine its quality. Clearly, this vector formulates a corresponding ARMAX model with the ability to predict the output \hat{y}_t .

For parametric models there is a large number of predictor algorithms, whose functionality depends mostly on the kind of the selected model, as well as the occasional scope of prediction. For the ARMAX case, a well-suited, one step-ahead predictor is stated in (Ljung, 1999) and has the following form:

$$\hat{y}_{t+1} = \frac{B(q)}{C(q)} \cdot u_t + \left[1 - \frac{A(q)}{C(q)} \right] \cdot y_t \quad (19)$$

In Eq. (19) the predictor can be viewed as a sum of filters, acting upon the data set and producing model's output at time $t+1$. Both of these filters have the same denominator dynamics, determined by $C(q)$, and they are required to be stable, in order to predict stable outputs. This is achieved by letting the roots of $C(q)$ have magnitude greater than one, requirement which coincides with the invertibility property of $H(q)$ transfer function (see Sec. 2).

4.2 Choice of objective function

The choice of an appropriate objective function performs vital role in any optimization problem. In most of the cases the selection is problem-oriented and, despite its importance,

this topic is very often undiscussed. However, in optimization theory stands as the starting point for any numerical algorithm, deterministic or stochastic, and is in fact the tool for transmitting any given information of the test case into the algorithm, in a way that allows functionality. For the ARMAX parameter estimation problem, the objective function that has been designed lies in the field of quadratic criterion functions, but takes a slightly different form, which enforces the adaptivity of the hybrid optimization algorithm to the measured data.

The objective function (of) is computed from the following pair of equations

$$of = v_N^* = 100 - fit \quad (20)$$

where

$$fit = 100 \cdot \left\{ 1 - \frac{\sum |\hat{\epsilon}_t(1/p)|}{\sum |y_t|} \right\} \quad (21)$$

Equation (21) can be considered as the ratio of the absolute error integral to the absolute response integral. When fit reaches the value 100, the predicted time-series is identical with the measured one. In this case, of results to zero, which is the global minimum point. Nevertheless, it must be noted that for a specific parameter vector, the global minimum value of the corresponding objective function is not always equal to zero, since the selected ARMAX structure may be unable to describe the dynamics of the true system.

The proposed method, as already mentioned, guarantees the stability of the estimated ARMAX model, by penalizing the objective function when at least one root of $A(q)$ or $C(q)$ polynomials lies within the unit circle of the complex plane. Thus, the resulted models satisfy the required conditions stated in Sec. 2.

4.3 Model order selection

The selection of a specific model among a number of estimated ones, is a matter of crucial importance. The model which shall be selected for the description of the true system's dynamics, must have as small over-determination as possible. There is a large number of statistical tests that determine model order selection, but the most common are the Residual Sum of Squares (RSS) and the Bayesian Information Criterion (BIC).

The RSS criterion is computed by a normalized version of (9), that is,

$$RSS = \frac{\|\hat{\epsilon}_t(1/p)\|}{\|y_t\|} \quad (22)$$

where $\|\cdot\|$ denotes the Euclidian norm. The RSS criterion generally leads to over-determination of model order, as it usually decreases for increasing orders. The BIC criterion overcomes this fact by penalizing models with relatively high model order

$$BIC = \ln \left[\frac{RSS}{N} \right] + \dim(\hat{\theta}) \cdot \frac{\ln(N)}{N} \quad (23)$$

Clearly, both of the methods indicate the "best" model, as the one minimizing (22) and (23) respectively.

5. Implementation of the method

In this Section the proposed methodology is implemented to the identification of a testing apparatus described below, by means of SISO ARMAX models. The process of parametric modelling consists of three fundamental stages: In the first stage, the delay nk shall be determined, in the second stage ARMAX(na, nb, nc, nk) models will be estimated, using the method described in Sec. 4, while in the third, the corresponding (selected) ARMAX model will be further examined and validated.

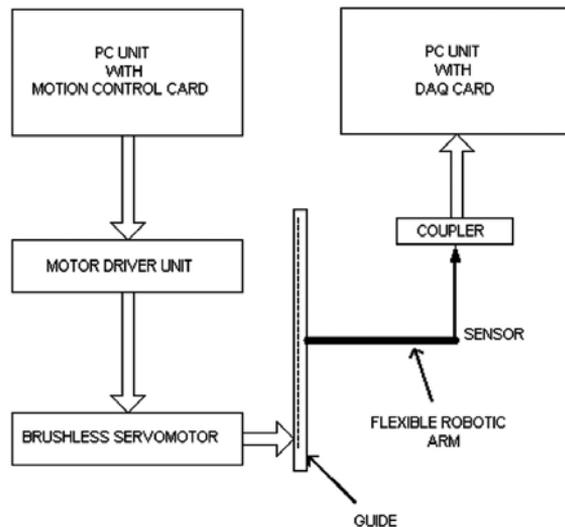


Figure 2. The testing apparatus

5.1 The testing apparatus

The testing apparatus is presented in Fig. 2. A motion control card, through a motor drive unit, which controls a brushless servomotor, guides a flexible robotic arm. A piezoelectric sensor is mounted on the arm's free end and acquires its transversal acceleration, by means of a DAQ device. The transfer function, considered for estimation in this study, is that relating the velocity of the servomotor with the acceleration of the arm. The velocity signal selected to be a stationary, zero-mean white noise sequence. The sampling frequency was 100 Hz and the number of recorded data was $N = 5000$, for both input and output signals.

5.2 Post-treatment of data

The sampled acceleration signal found to be strongly non-stationary, with no fixed variance and mean. Thus, stationarity transformations made before the ARMAX estimation phase. Firstly, the BOX-COX (Box et al., 1994) transformation was used with $\lambda_{BC} = 1.1$, for the stabilization of variance, and afterwards difference transformations with $d = 2$ for the stabilization of mean value were implemented. The resulted acceleration signal, as well as the velocity one, was zero-mean subtracted. The final input-output data set is presented in Fig. 3. For the estimation of acceleration's spectral density, Thompson's multi-paper method [23] has been implemented, with time-bandwidth product $n_T = 4$, and number of Fast Fourier Transforms $N_{FFT} = 2^{13}$. The estimated spectral density is presented in Fig. 4. Clearly, there are

three spectral peaks appearing in the graph, corresponding to natural frequencies of the system at about 2, 5 and 36 Hz, while an extra area of natural frequency can be considered at about 17 Hz. An additional inference that can be extracted, is the high-pass trend of the system.

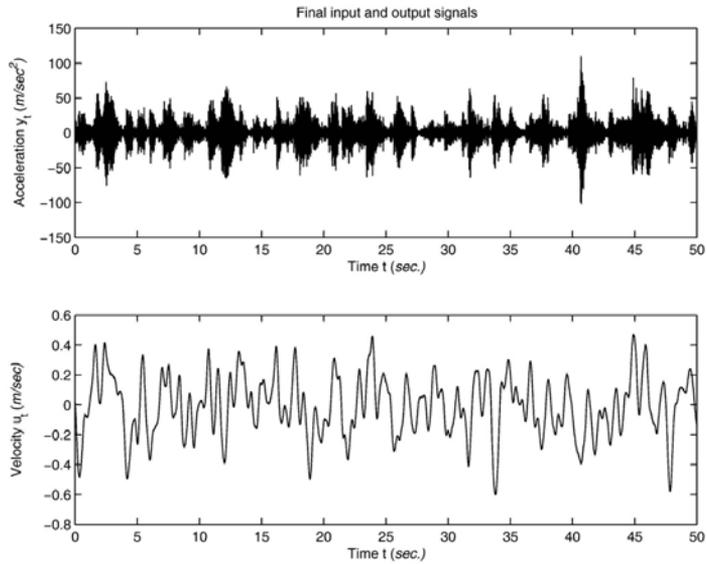


Figure 3. The input-output data set

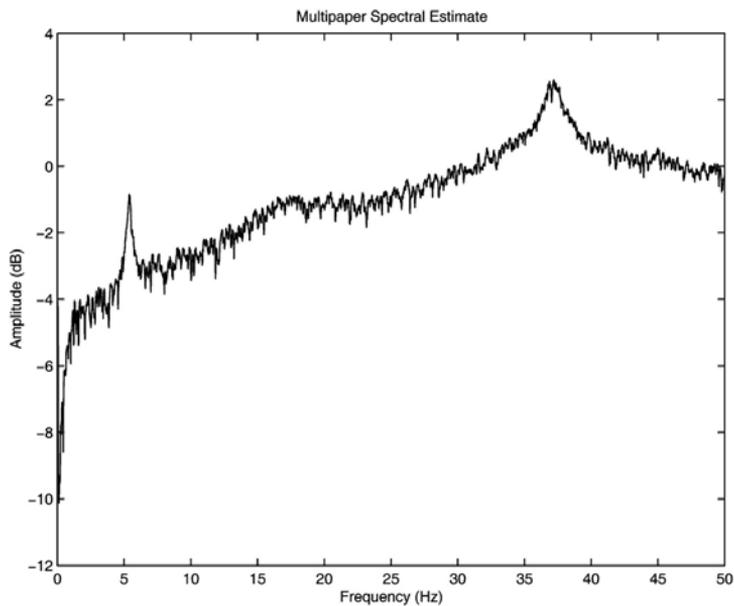


Figure 4. Acceleration's estimated spectrum

For the subsequent tasks, and after excluding the first 500 points to avoid transient effects, the data set was divided into two subsets: the estimation set, used for the determination of

an appropriate model by means of the proposed method, and the validation one, for the analysis of the selected ARMAX model.

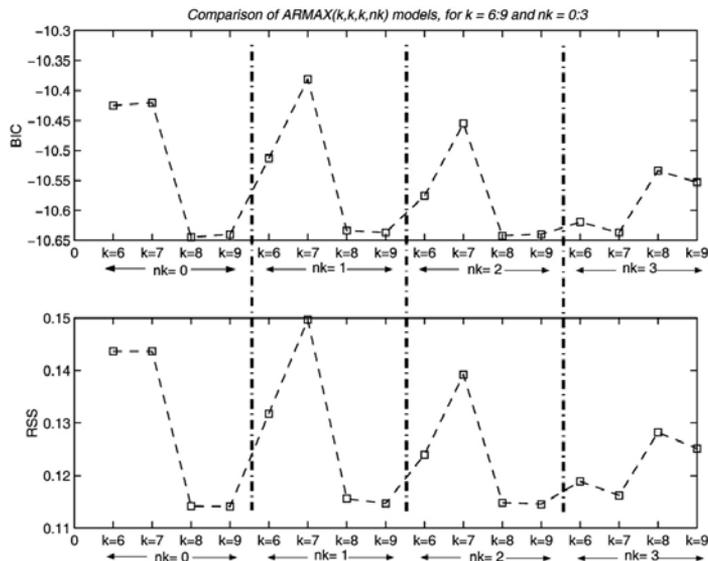


Figure 5. Selection of system's delay

5.3 Determination of the delay

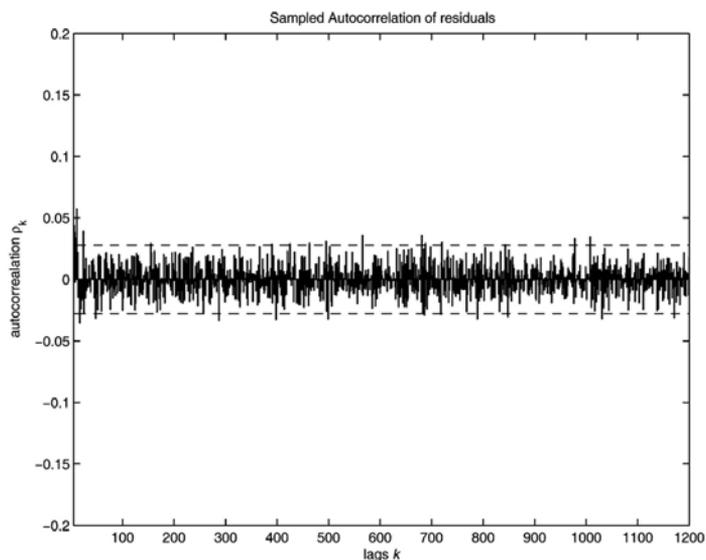


Figure 6. Autocorrelation of residuals

For the determination of delay, ARMAX(k, k, k, nk) models were estimated, with $k = 6, 7, 8, 9$ and $nk = 0, 1, 2, 3$. The resulted models were evaluated using the RSS and BIC criterions. Figure 5 presents the values of the two criterions, with respect to model order, for the various delays. The models with delay $nk = 3$ presented better performance and showed

smaller deviations between them. Thus the delay of the system was set to $nk = 3$. This is an expected value, due to the flexibility of the robotic arm, which corresponds to delayed acceleration responses in its free end.

5.4 Estimation of ARMAX($na, nb, nc, 3$) models

The selection of an appropriate ARMAX model, capable of describing input-output dynamics, and flexible enough to manage the presence of noise, realizes through a three-phase procedure:

- In the first phase, ARMAX($k, k, k, 3$) models were estimated, for $k = 6 : 14$. The low bound for k is justified from the fact that the three peaks in the estimated spectrum (see Fig. 4), correspond to three pairs of complex, conjugate roots of $A(q)$ characteristic polynomial. The upper bound was chosen in order to avoid over-determination. The resulted models qualified via BIC and RSS criteria and the selected model was ARMAX(7, 7, 7, 3).
- The second phase dealt with the determination of $C(q)$ polynomial. Thus, ARMAX(7, 7, $k, 3$) models were estimated, for $k = 2 : 16$. Again, BIC and RSS criteria qualified ARMAX(7, 7, 7, 3) as the best model, and also the one with the lowest variance of residuals' sequence.
- In the third stage ARMAX(7, $k, 7, 3$) models were estimated for the selection of $B(q)$ polynomial. Using the same criteria, the ARMAX(7, 6, 7, 3) model was finally selected for the description of the system.

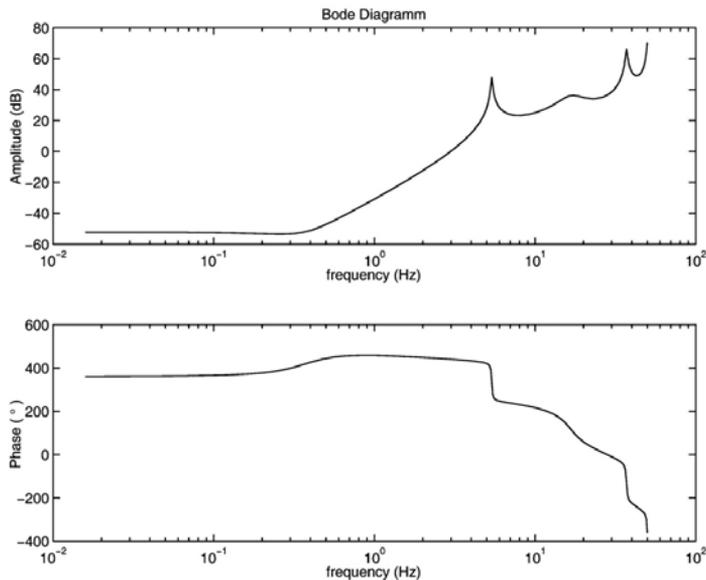


Figure 7. Model's frequency response

The implementation of the proposed methodology in the above procedure presented satisfying performance, as the hybrid optimization algorithm presented quick convergence rate despite model order, the resulted models were stable and invertible and over-determination was avoided.

5.5 Validation of ARMAX(7,6,7,3)

For an additional examination of the ARMAX(7, 6, 7, 3) model, some common tests of its properties have been implemented. Firstly, the sampled autocorrelation function of model residuals was computed for 1200 lags and it is presented in Fig. 6. It is clear that, except few lags, the residuals are uncorrelated (within the 95% confidence interval) and can be considered white. In Fig. 7, the frequency response of the transfer function $G(q)$ (see (6)) is presented. The high-pass performance of the system is obvious and coincides with the same result that was extracted from the estimated spectral density in Fig.4. Figure 8 displays a simulation of the system, using a fresh data set that was not used in the estimation tasks (the validation set). The dash line represents model's simulated acceleration, while the continuous line the measured one.

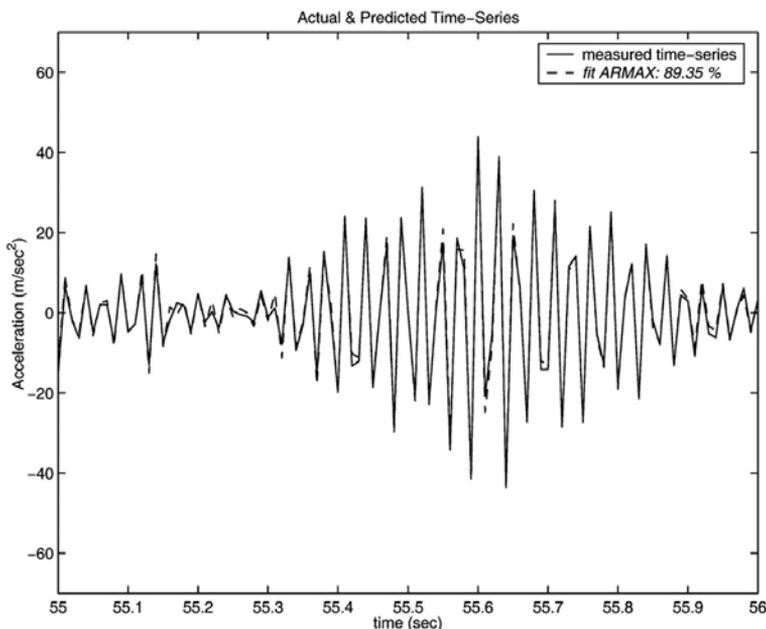


Figure 8. One second simulation of the system

Finally, in Table 2 the natural frequencies in Hz and the corresponding percentage damping of the model are presented. While the three displayed frequencies were detected successfully, the selected model was unable to detect the low frequency of 2 Hz, probably due to its high-pass performance. Yet, this specific frequency was unable to be detected, even from higher order estimated models.

Poles	$\omega_n(\text{Hz})$	$\zeta(\%)$
$0.94 \pm 0.33i$	5.36	0.91
$0.43 \pm 0.71i$	16.65	17.50
$-0.67 \pm 0.70i$	37.00	1.07

Table 2. Natural frequencies and corresponding damping

6. Conclusion

In this paper a new method for the estimation of SISO ARMAX models was presented. The proposed methodology lies in the context of Evolutionary system identification. It consists of a hybrid optimization algorithm, which interconnects the advantages of its deterministic and stochastic components, providing superior performance in PEM, as well as a two-stage estimation procedure, which yields only stable models. The method's main characteristics can be summarized as follows:

- improvement of PEM is implemented through the use of a hybrid optimization algorithm,
- initial "guess" is not necessary for good performance,
- convergence in local minima is avoided,
- computational complexity is sufficiently decreased, compared to similar methods for Evolutionary system identification. Furthermore, the method has competitive convergence rate to conventional gradient-based techniques,
- stability is guaranteed in the resulted models. The unstable ones are penalized through the objective function,
- it is successive, even in the presence of noise-corrupted measurement.

The encouraging results suggest further research in the field of Evolutionary system identification. Specifically, efforts to design more flexible constraints are taking place, while the implementation of the method to Multiple Input-Multiple Output structures is also a topic of current research. Furthermore, the extraction of system's valid modal characteristics (natural frequencies, damping ratios), by means of the proposed methodology, is an additive problem of crucial importance.

Evolutionary system identification is an growing scientific domain and presents an ongoing impact in the modelling of dynamic systems. Yet, many issues have to be taken under consideration, while the knowledge of classical system identification techniques and, additionally, signal processing and statistics methods, is necessary. Besides, system identification is a problem-specific modelling methodology, and any possible knowledge of the true system's performance is always useful.

7. References

- Baeck, T; (1996). *Evolutionary Algorithms in Theory and Practice*. New York, Oxford University Press.
- Box, GEP; Jenkins, GM; Reinsel, GC; (1994). *Time series analysis, forecasting and control*. New Jersey, Prentice-Hall.
- Billings, SA; Mao, KZ; (1998). Structure detection for nonlinear rational models using genetic algorithms. *Int. J. of Systems Science* 29(3), 223-231
- Dorigo, M; Bonabeau, E; Theraulaz, G; (2000). Ant algorithms and stigmergy. *Future Generation Computer Systems* 16, 851-871.
- Fleming, PJ; Purshouse, RC; (2002). Evolutionary algorithms in control systems engineering: a survey. *Control Eng. Practice* 10(11), 1223-1241.
- Fletcher, R; (1987). *Practical Methods of Optimization*. Chichester, John Wiley & Sons.
- Gray, GJ; Murray-Smith, DJ; Li, Y; Sharman, KC; Weinbrenner, T; (1998). Nonlinear model structure identification using genetic programming. *Control Eng. Practice* 6, 1341-1352

- Kanarachos, A; Koulocheris, D; Vrazopoulos, H; (2002). Acceleration of Evolution Strategy Methods using deterministic algorithms. *Evolutionary Methods for Design, Optimization and Control*, In: *Proc. of Eurogen 2001*, pp. 399–404.
- Kristinsson, K; Dumont, GA; (1992). System Identification and Control using Genetic Algorithms. *IEEE Trans. on Systems, Man, and Cybernetics* 22, 1033–1046.
- Ljung, L; (1999). *System Identification: Theory for the User*. 2nd edn., New Jersey, Prentice-Hall.
- Monmarche, N; Venturini, G; Slimane, M; (2000). On how *Pachycondyla apicalis* ants suggest a new search algorithm. *Future Generation Computer Systems* 16, 937–946.
- Mrad, RB; Fassois, SD; Levitt JA, Bachrach, BI; (1996). On-board prediction of power consumption in automobile active suspension systems – I: Predictor design issues, *Mech. Syst. Signal Processing*, 10(2), 135–154.
- Mrad, RB; Fassois, SD; Levitt, JA; Bachrach, BI; (1996). On-board prediction of power consumption in automobile active suspension systems – II. Validation and performance evaluation. *Mech. Systems and Signal Processing* 10(2), 155–169.
- Nocedal, J; Wright, S; (1999). *Numerical Optimization*. New York, Springer-Verlag
- Oppenheim, AV; Schaffer, RW; (1989). *Discrete-time signal Processing*. New Jersey, Prentice-Hall .
- Percival, DB; Walden, AT; (1993). *Spectral Analysis for Physical Applications: Multipaper and Conventional Univariate Techniques*. Cambridge, Cambridge University Press.
- Petsounis, KA; Fassois, SD; (2001). Parametric time domain methods for the identification of vibrating structures – A critical comparison and assessment. *Mech. Systems and Signal Processing* 15(6), 1031–1060.
- Rajesh, JK; Gupta, SK; Rangaiah, GP; Ray, AK; (2001). Multi-objective optimization of industrial hydrogen plants. *Chemical Eng. Sc.* 56, 999–1010.
- Rodriguez-Vazquez, K; Fonseca, CM; Fleming, PJ; (1997). Multiobjective genetic programming: A nonlinear system identification application. In: *late breaking papers at the 1997 Genetic Programming Conf.* 207–212 .
- Schoenauer, M; Sebag, M; (2002). Using Domain knowledge in Evolutionary System Identification. *Evolutionary Methods for Design, Optimization and Control*, In: *Proc. of Eurogen 2001*, 35–42 .
- Schwefel, HP; (1995) *Evolution & Optimum Seeking*. New York, John Wiley & Sons Inc.
- Soderstrom, T; Stoica, P; (1989). *System Identification*. Cambridge, Prentice Hall Int.
- Tan, KC; Li, Y; (2002). Grey-box model identification via evolutionary computing. *Control Eng. Practice* 10(7), 673–684.

Evolutionary Computation of Multi-robot/agent Systems

Philippe Lucidarme

*University of Angers, Engineering Systems Laboratory (LISA)
France*

1. Introduction

Evolutionary computation (EC) and multi-agent systems (MAS) are two research topics offering interesting similarities. Both are biologically inspired; Evolutionary computation takes its origins from genetic algorithms inspired by the Darwinian principle (Goldberg 1989) and multi-agent systems are inspired from ethological studies, mainly ant colonies. Both are based on a population, made of individuals (EC) or agents (MAS). Evolutionary computation is based on the iterative development of a population. Each population is made up of individuals partially selected from the previous generations. In the case of multi-agent systems the population is made of agents. The exact definition of what is an agent is highly dependant on the context. In robotics systems, agents are usually described as a robot or part of a robot. In the first case, the multi-agent system (or multi-robots system) is composed of several robots interacting or acting together. In the second case, a unique robot is composed of several agents, each managing an entity (an actuator, a sensor ...). In both cases, multi-agent systems offer a breaking down of a complex problem into several simpler tasks. The last (but not least) common point between evolutionary computation and multi-agent systems is the similarity between emergence and evolution; previous works (Brooks, 1986; Arkin 1992; Drogoul & Ferber, 1992) have shown that a population is able to perform tasks that an isolated agent is not able to do. A similar phenomenon appears in evolutionary computation: it's the combination of several individuals that allows the increase of fitness throughout the generations.

2. Related works

Multi-agent systems have been studied for many years and the following definitions used in the rest of the chapter are accepted by the MAS community and described in this section. All of them are fully detailed and justified in (Ferber 1999).

2.1 Distributed and supervised multi-agent systems

Multi-agent systems can be classified into two categories: supervised or distributed systems. In robotics, most of the system may be viewed as supervised: a central agent (the computer or processor) gathers information (from sensors) and sends commands to the other agents (the actuators). The main drawback of such architecture is that a failure on the central agent

may be fatal to the whole system. On the contrary, in distributed systems, agents are as much as possible autonomous and the loss of agents may be supported by the system.

2.2 Communication between agents

Previous works (Drogoul 1992; Tucker 1994) have shown that communication is a key point in multi-agent systems. Such systems are classified in 3 categories: without communication, with implicit communication and with explicit communication. When no communication isn't used at all, the agents are fully independent. Such systems are generally limited in term of cooperation. When the agents are communicating without using classical data transmissions, it is called implicit communications. Famous examples are the pheromones in ant colonies. To find their way back or to inform the rest of the colony from danger, the ants use pheromones which act as messages. In robotics, some example may be found based on implicit communication. For example, at the European cup of robotics in 2006, the purpose of the game was to gather and sort coloured balls on a playground. A team of student built a robot composed of several independent modules. The communication between modules was done using the presence or not of balls inside the module. The simplicity of this implicit communication based system allowed a high reliability, essential in these kind of challenge. But most of the robots use explicit communications, for example bus protocols like I²C, BUS CAN or TCP IP are used to communicate between the different parts of a robot and radio frequencies or infrared signals are used to communicate between robots.

2.3 Heterogeneity in multi-agent systems

Heterogeneity is used in multi-agent systems to evaluate the difference between agents. When all the agents are physically identical and controlled by the same rules, the system is said to be highly homogeneous. On the other hand, when each agent is specialized or dedicated to a task, the system is said to be highly heterogeneous.

2.4 Multi-agent systems and evolutionary computation

An evolutionary algorithm may be considered as a particular case of homogeneous multi-agent system where agents are individuals. The rules governing these agents are crossovers and mutations. Crossovers may be considered as a supervised rule: a central computer gathers the fitness and the artificial chromosomes of the whole population before computing the next generation. Mutations may be viewed as distributed rules because the mutation of a chromosome is not dependant from the rest of the population. Almost all the previous works, applying evolutionary computation to robots and generally inspired by the works of Dario Floreano (Floreano & Mondada 1994; Nolfi & Floreano 2000), were based on this principle.

This chapter will focus on the combination between EC and MAS to create controllers for autonomous robots. According to the previous observations, the experiments presented in this chapter take advantage of both systems. Agents are based on reactive controllers allowing fast responses and reducing computation time. The presented systems are fully distributed.

In the first part of the chapter, an experiment shows that a genetic algorithm can be fully distributed into a group of real mobile robots (Lucidarme 2004). The second experiment, made on a humanoid robot (HRP2), will describe how a unique robot can be seen as an evolutionary optimized multi-agent system.

3. Fully distributed evolutionary robots

3.1 Hypotheses

In this multi-robot experiment, each robot is considered as an agent. A homogeneous population is considered, i.e. all the robots have the same capabilities of sensing, acting, communicating and processing. The system is fully distributed, i.e. information about the agents are never centralized. Communications between agents are explicit. Moreover, the considered task is safe and robust reactive navigation in a clustered environment for exploration purposes. The robots are programmed at the beginning of the experiment neither for obstacle avoidance nor for enlarging the explored area, and nor for executing more complex actions. On the contrary, the agents have to find by themselves an efficient policy for performing the obstacle avoidance tasks. The on-line self-learning procedure is based upon the principles of evolutionary algorithms that allow a faster convergence rate than the classical learning algorithms, as it is shown in the following. The algorithm's operation is then illustrated by considering the exploration problem, which is simple to evaluate and to implement on real mobile robots. The population size is constant.

3.2 Evolution of physical robots

Type 1 (Lucidarme & Simonin 2006) is a small mobile robot (shown on figure 1) with a diameter of 13 cm and a weight of 800 g (including batteries) used in this first experiment. It has many of the characteristics required by the evolutionary approach to autonomous robot learning. The robot is fully autonomous; an embedded PC (80486 DX with 66 MHz clock) manages the robot. Control to sensors and actuators is transmitted by the PC104 bus. Two wheels actuate it. Two small passive ball-in-socket units ensure the stability. DC motors equipped with incremental encoders (352 pulses per wheel's revolution) control the wheels. The encoders are used for speed control but also to measure the performance index detailed in the next sections. The robot is surrounded with 16 infrared emitters and 8 receivers. The sensors use a carrier frequency of 40 kHz for a good noise rejection. These sensors are also used to communicate between agents. A communication protocol has been developed to differentiate obstacles from other robots: if a signal is received when the robot has stopped emitting for a short period, this means that another agent is close to it.

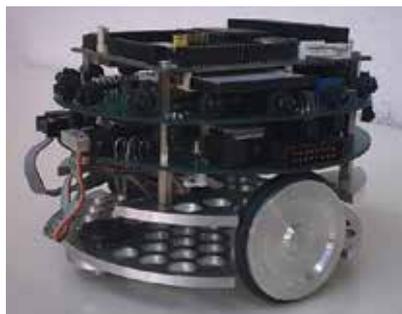


Figure 1. Picture of the mobile robot Type 1

3.3 Chromosome encoding

The controller of the robot is inspired from reinforcement learning (based on Markovian processes) where the policy is described by the association between states and actions. Such

controller allows the comparison with other learning approaches. The inputs of the sensorimotor system are composed of five states listed in Table 1.

State 1	No obstacle
State 2	Left obstacle
State 3	Right obstacle
State 4	Front obstacle
State 5	Robot is jammed

Table 1. Different states of an agent's sensory system

These states can easily be recognized by the proximity sensors of any robot. Of course, more inputs would be available if fuzzy processing was applied. On the other hand of the control system are the elementary behaviors of the robots given in Table 2.

Behavior 1	Go forward
Behavior 2	Turn right
Behavior 3	Turn left
Behavior 4	Go backward

Table 2. The set of elementary actions

Actions actuate immediately the wheel motors. An individual of the population considered for the evolutionary learning is the list of connections between inputs and outputs. It encodes thus the "synapses" of the robot's sensorimotor control system. Such states and elementary actions have been chosen to guarantee that the behavior is learnable during the battery life. The chromosome of a robot is the concatenation (a string) of N words in a subspace of the $\{0,1\}^M$ space. N is the number of inputs, and M the number of outputs. Of course, each word contains a single bit '1' (Table 3). The population of robots will evolve following the evolution of the embedded chromosomes under the action of genetic operators.

0	0	0	1	1	0	0	0	0	1	0	0	1	0	0	0	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Table 3. An example of a chromosome string

3.4 Operators and rules

Initialization: at the beginning of the experiment, the various chromosomal strings are filled at random or with the same behavior, like a string of "go ahead". For the simulations and experiments reported in this paper, the strings have been randomly generated to increase the diversity of the population.

Fitness: in many learning techniques, some kind of supervisor exhibits templates, and gives a rating to the agent's resulting behavior. In the case of genetics algorithms this upper level may also evaluate the fitness of each individual of the population with respect to a required performance. The individuals are then ranked (the selection operation) before applying the genetic operators. As we are looking here for a fully distributed evolution, a capability of local self-evaluation is given to each robot, but it is never conscious of the global population efficiency. However, following the general principles of self-learning algorithms, each individual (here the agent numbered i) computes its own current fitness using equation 1.

$$R_{N(i)}(i) = (1 - \alpha(i))R_{N(i)-1}(i) + \alpha(i)F_{N(i)}(i) \quad (1)$$

Where

$$\alpha(i) = \frac{1}{1 + N(i)}$$

$N(i)$ is the number of time steps since the beginning of the estimation by agent i

$R_{N(i)}(i)$ is the estimated reward at time $N(i)$

$F_{N(i)}(i)$ is the instantaneous reward at time $N(i)$

In this obstacle avoidance problem, the instantaneous reward is computed as being the forward distance traveled during an elementary time step. This distance is computed using the motor's encoders. Thus, the current reward resulting from applying the current policy (the chromosome) is the average distance since initialization or since the last change of the agent's chromosome by crossover or mutation. Such a computation automatically penalizes too numerous turns and reverse motions.

Crossover: most of the solutions proposed by the others investigators call for global communication between the agents. In this way, the classical genetic algorithm technique is used: selection by considering the whole population, then crossover of two chromosomes at a periodic average rate. This method suffers from the lack of parallelism since only two agents can be concerned at the same time. It even may require a slackening of the robot moves to allow the complex communication, agent recognition and signal processing. In order to avoid these major drawbacks, the solution proposed here uses a local and simple communication. This way, any pair of robots meeting each other may perform crossover. The formal conditions are the following: the robot-to-robot distance is short enough to communicate and both robots have not recently performed a crossover or mutation operation. The first condition ensures possible parallelism, while the second prevents any robot from changing its current policy before having evaluated it over a significant number of steps. When crossovers are complete, agents i and j have new chromosomes, i.e. policies, according to the probabilities given by equation 2.

$$P(i) = \frac{R_{N(i)}(i)}{R_{N(i)}(i) + R_{N(j)}(j)} \quad (2)$$

Crossovers are not sufficient to ensure the convergence of the system towards the best solution, especially when the population size is small; the optimal chromosome may not be dispatched in the initial population preventing crossovers from creating this optimal chromosome string. In that case, the agents may be trapped by a local maximum, and the population no longer evolves from that common state. As usual, mutations are necessary to escape from local extrema and to explore a wide domain of the chromosomal state space.

Mutation: in the classical genetic algorithms, mutations are performed at random. In our application, this could be an important drawback since we are looking for on-line learning. It cannot be admitted that a robot changes its policy to a far less efficient one and waits for a long time before re-improving it by a new mutation or by crossover. To solve these problems, the formal conditions for mutation are adopted: the agent has not performed a previous mutation recently and the agent's fitness is low. Notice that again such a method allows multiple simultaneous mutations of several agents. As for crossover, the first condition ensures that the robot has had enough time for computing a good estimation of its

own fitness. The second condition prevents from changing good policies into worse ones. The adopted probability of mutation is dependent on the individual performance index. It ensures that the worst policies are more likely to mutate, while the best ones possess a non-zero probability for escaping a possible local maximum.

3.5 Simulation results

The algorithm was first evaluated via Matlab allowing simulations with a large number of robots. The moving agents never compute their absolute position and orientation. The knowledge of these variables only serves for the displays. On the contrary, the real sensory system is simulated and provides the agent with its local perception. The real wheel control is also simulated. An example of display is shown on Figure 2.

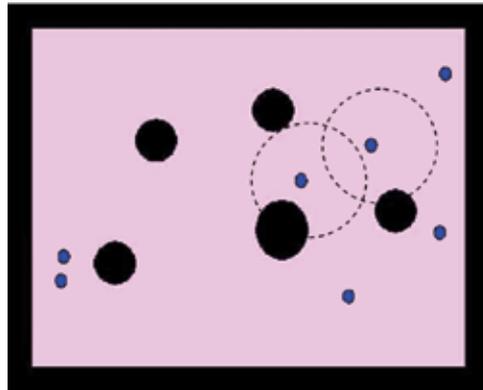


Figure 2. Snapshot of the simulator with 7 robots (small circles) and circular obstacles (in black)

During a simulated elementary sensorimotor cycle of time, each agent updates its current state, and either continues its current policy or performs mutation (if allowed) or crossover (if possible). To make crossover possible, the sensory protocol differentiates another agent from a passive obstacle, and is able to compute the robot-to-robot distance. The two dashed circles surrounding two robots in Figure 2 show the sensor ranges. When a crossover is allowed, it takes several cycle times due to the need for communicating information. Otherwise, each of the two robots considers the other as an obstacle.

The first step is to set the simulation parameters. The communication range is measured on real robots and scaled to the simulated environment. Then it is necessary to adjust the coefficients in the probability function for mutations. The answer was found thanks to simulations. If the delay is short, the mutation occurs frequently. The search state is explored very quickly, but the performance estimation is erroneous. On the contrary, if the delay is long, the space state is explored very slowly, but the estimation is very good. The delay between two crossovers was also studied. The conclusions are close to those of mutations. If the delay is too short, robots will always crossover with the same agent. On the contrary, if the delay is too long, it takes a lot of time to explore the policy space state. The minimal number of cycles is chosen to fulfill the following condition: two agents having the optimal policy will not meet twice successively. Here about 300 cycles are necessary.

Figure 3 shows the end of a simulation. The first observation is the explored part of the environment (in white). It results from the obtained emergent behavior due to the reactivity

of the multi-agent system. We have also checked that the optimal solution, given in Table 3, is always reached whatever the initial conditions and the shape of the environment.

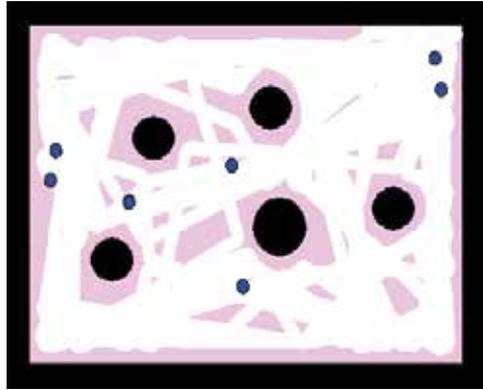


Figure 3. Snapshot of the simulator at the end of an experiment

At the end of the experiment, the chromosome encodes the optimal sequence (table 3).

1: No obstacle	1: Go forward
2: Left obstacle	2: Turn right
3: Right obstacle	3: Turn left
4: Front obstacle	2 or 3 : Turn left or right
5: Robot is jammed	4: Go backward

Table 3. Different states of an agent's sensory system

In most cases, the observed evolution is the following:

1. At the beginning, the agents try random strategies. A collision quickly occurs, and the robot is jammed. The average traveled distance drops very fast, and a mutation occurs soon. Mutations stop when the behavior 4 (go backward) is associated to the state 5 (robot is jammed).

2. The agents are free in the environment, and one of them is likely to associate the first behavior (go forward) to the first state (no obstacle). It generally travels a long distance in the environment and propagates its chromosomal string to other agents by crossover.

3. As many agents move fluently in the environment, more crossovers occur. The optimal policy is thus given to at least one agent. Its own fitness grows quickly.

4. As soon as such a robot performs the best sequence, it travels all over the environment and meets more agents than the other ones. By doing so, it transmits its string to many robots and all the population quickly performs the best behavior. The non-null mutation rate imposed in "good" populations ensures that the absolute maximum is obtained.

Another important parameter is the size of the population, i.e. the number of robots. Intuitively, the more agents there are, the faster the best solution is found. This hypothesis is true, if the condition of convergence is: "at least one agent finds the optimal sequence". We decided to stop the simulations when each individual had performed the chromosomal sequence described in table 3. For each number of robots, twenty simulations were performed and the average number of cycles was recorded. The results are shown on Figure 4.

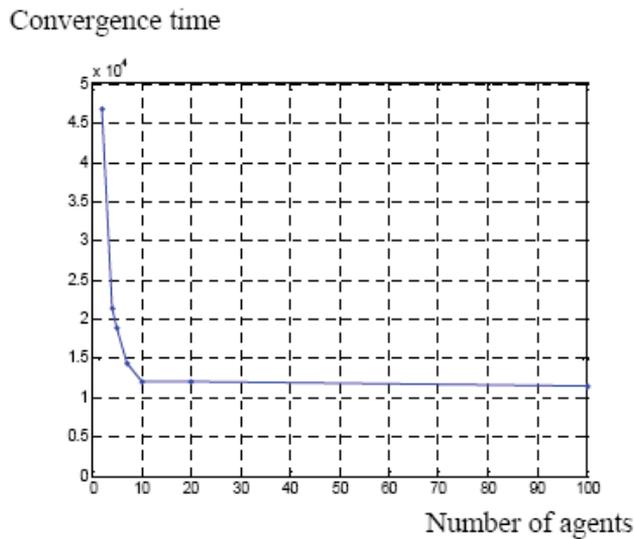


Figure 4. Convergence time versus number of robots

It can be seen on figure 4 that, when less than ten robots are used, the average convergence time decreases with the number of robots as in classical evolutionary algorithms. With more than 10 robots, the convergence time becomes constant. The explanation is that when the number of agents is high, the best behavior is quickly found (because the search space is quickly explored). But the time to propagate this optimal chromosomal string into the whole population is longer when the population is big, this phenomena counterbalances the fast exploration of the state space.

3.6 Experimental results

The experimental environment is shown on Figure 5. It is 4.80 m long and 3.60 m wide. Four autonomous robots are used in the experiments, and obstacles can be moved, added or suppressed. The maximum speed of the robots is 1 m/s but the speed has been limited to .3 m/s to prevent eventually violent collisions. The sensorimotor cycle time is about 15 ms. The range of the infrared system is typically of .5 m: it ensures here the required compromise between the needs for enough crossovers and for safety with respect to collisions. Following these numerical data and the experience gained during the simulations, the minimum time between two successive crossovers has been set to 3 seconds.

The first conclusion of this real experiment is the fact that the algorithm always converge to one of the two optimal solutions in a reasonable time; less than ten minutes for the simple obstacle avoidance task (5 states and 4 actions). This real experiment has also confirmed the emergence of an important phenomenon: when a robot performs an efficient strategy, it naturally increases its chance to be in the next generation i.e. it travels smoothly in the environment and increases its chance to meet the others robots; he becomes a more popular candidate for reproduction.



Figure 5. Distributed evolution of real robots

3.7 Conclusion

This first experiment has proved the possibility of transposing evolutionary computation into a real group of robots. Experimental results have shown that this solution takes effectively advantages (fault tolerance and emergence) from multi-agent systems as explained in the introduction. As the system is fully distributed it becomes fault tolerant; for example, if a robot is jammed or broken, the rest of the population still works. During the experiments, robots has been removed and added into the population without any trouble. Having a group of robots decreases the learning duration because the search space is more quickly explored compared to an equivalent experiment made on a unique robot (Floreano & Mondada 1994). But having a population of robots may also have drawbacks; the size of the environment must be proportional to the number of robots. Simulation results have shown that, over a threshold, increasing the population size doesn't decrease the convergence time in regards with the size of the environment. Using at the same time several complex or expensive robots is not always possible; i.e. using ten humanoid robots at the same time to process learning is not actually feasible. The next section has been motivated by this conclusion and will describe an experiment where a unique robot is seen as an evolutionary optimized multi-agent system.

4. Evolutionary optimized multi-agent system

4.1 Hypotheses

As explained previously, a unique robot may be structured as a multi-agent system where each part of the robot is a communicating and acting agent. The task considered here is the tracking of a target with the hand of a humanoid robot without falling or being unbalanced. We assume that the target is reachable by the robot without walking, i.e. feet are considered as linked on the floor and the target is included in the workspace of the body. The position of the target is assumed to be known by the robot; it may be done by using the vision system of the robot for example. The body of the robot is decomposed in agents each controlling a joint. The system is fully distributed, i.e. information about the agents are not centralized. Of course existing robots use a central processor and have not been designed to support such architecture. Currently, the distribution is simulated by sharing the central processor, but it may be possible to embed a controller in each joint to perform the agent behavior. Moreover, it is actually done on existing robots (Murase and al. 2001) for controlling the

joints with a PID. To allow this distribution of the system, each agent only communicates with its neighbors.

A homogeneous population is considered, however, as each agent controls a joint of the humanoid robot, the computed model and parameters vary. As the robot has a human shape, it may be a plus to provide natural (human-like) motions. An evolutionary strategy is used to optimize the parameters of the system. The purpose of the presented experiments is to show that after the evolution, the system automatically manages redundancy and failures on actuators.

4.2 Description of the architecture

The robot considered in the following is HRP2. This humanoid robot has 30 degrees of freedom (6 by leg, 6 by arm, 1 by hand, 2 for the hip and 2 for the neck). This robot has been chosen for its high redundancy, even if many other robots may have been used.



Figure 6. The humanoid robot HRP2

In the proposed distributed approach each actuator q_i is respectively associated with the agent A_i . Each joint or agent acts independently from the other joints in order to minimize the Euclidian distance ε between the hand and the target, i.e. reaching the target with the hand. Each agent is described by the following 3 items : input, output and behavior .

Input : information or data, to which the agent can access. The agent A_i knows the following variables:

- ${}^0T_{i-1}$: transformation matrix linking the original frame F_0 (based on the ground) to the current joint q_{i-1} . This information is communicated by the agent A_{i-1} except for the first agent.

- ${}^i T_n$: transformation matrix linking the joint q_{i+1} to the end effector. This matrix is communicated by the agent A_{i+1} except for the last agent.
- q_i : current measured position of the joint controlled by the agent A_i . This position is provided by a sensor on the joint (in the case of HRP2, an encoder provides the position of the joint)
- P_{Target} : coordinate of the target in the frame F_0 . We assume that this information is known by each agent. On a real physically distributed system, this information may be computed by a dedicated controller (based in the head of the robot for example) and transmitted to the closest agent. This agent transmits this information to the neighbors and the coordinates are propagated in the entire system.

Output: each agent A_i must provide two kinds of outputs: command on the actuator and information to the neighbors:

- Δq_i : command applied on the joint (angular speed).
- ${}^0 T_i$: transformation matrix linking the original frame F_0 (based on the ground) to the current joint q_i . This information is communicated to the agent A_{i+1} .
- ${}^{i-1} T_n$: transformation matrix linking the joint q_i to the end effector. This matrix is communicated to the agent A_{i-1} .
- P_{Target} : coordinate of the target transmitted to the agents A_{i-1} , A_{i+1} or both.

A general view of the architecture, showing the information exchanged between the agents is shown on figure 7.

Behavior: this is the way the agents link the input to the outputs. Note that, as the system is distributed, the behaviors are local, i.e. the agent A_i does not know the way the other agents will act. The global goal is to reach the target with the end-effector of the robot. Internal collisions are not considered here. The behavior of the agent A_i consists of computing at each time step the value Δq_i that minimizes the Euclidian distance between the position of the end effector P_n and the position of the target P_{target} in regard to the information known by the agent. Due to its efficiency and its simplicity, the gradient descent technique has been chosen. This choice was motivated by the following reasons:

- The mathematical relationship between the actuator's position and the distance to minimize is a known function. The derivative of this function may be easily computed.
- There are few parameters to set, just one for each agent and evolutionary computation is well suited for tuning these parameters.
- The main advantage is probably that the algorithm slides into the minima. It does not provide just the final solution, but provides a trajectory that slide from initial to final configuration. In the case of robots, we need more than the final position: we need a trajectory. Gradient descent provides both.

Each agent updates its output according to equations 3

$$\Delta q_i(t) = -\alpha_i \cdot \frac{d\mathcal{E}}{dq_i} \quad (3)$$

\mathcal{E} is the Euclidian distance between the hand of the robot and the target
 q_i is the current position of the joint i

α_i is a parameter that determine the behavior of the agent A_i . The influence of this parameter is described in the next section.

The derivative of the Euclidian distance is computed using the equation 4.

$$\frac{d\mathcal{E}}{dq_i} = \frac{(x_{Target} - x_n) \cdot \frac{dx_n}{dq_i} + (y_{Target} - y_n) \cdot \frac{dy_n}{dq_i} + (z_{Target} - z_n) \cdot \frac{dz_n}{dq_i}}{\sqrt{(x_n - x_{Target})^2 + (y_n - y_{Target})^2 + (z_n - z_{Target})^2}} \quad (4)$$

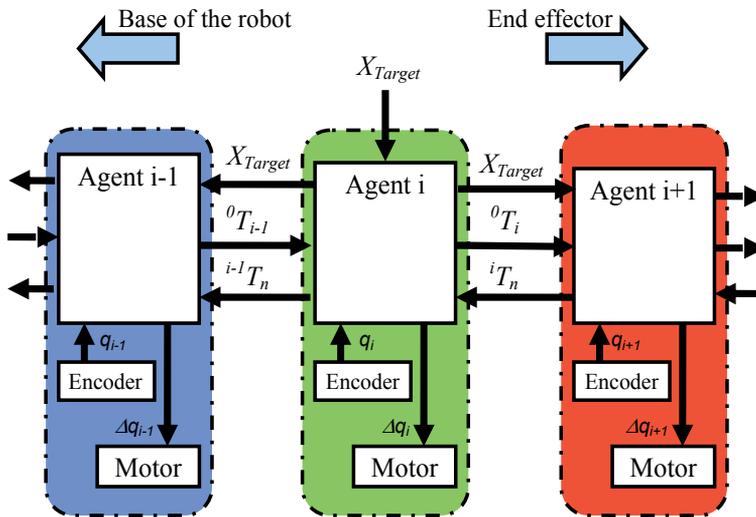


Figure 7. Overview of the multi-agent architecture

Note that, as the system is fully distributed, the agent A_i cannot compute $\frac{dx_n}{dq_i}$, $\frac{dy_n}{dq_i}$ and $\frac{dz_n}{dq_i}$

by using the jacobian because according to our hypothesis, this agent A_i cannot access to the other agent's dq . In spite of this, the matrix product described on equation 5 can compute the derivative of the end-effector position according to the joint i .

$$\frac{dP_n}{dq_i} = {}^0T_{i-1} \cdot \frac{d^{i-1}T_i}{dq_i} \cdot {}^iT_n \cdot X_0 \quad (5)$$

This relationship described on equation 4 is only true in the case of serial robots. Only one transformation matrix (${}^{i-1}T_i$) is expressed in term of q_i , t. The following terms can be considered as scalar and don't need to be derivated: dX_n/dq_i , ${}^0T_{i-1}$ and iT_n . The behavior of the agent A_i is described in the algorithm presented on the figure 8.

Algorithm AgentBehavior
input : ${}^0T_{i-1}, {}^i T_n, P_{\text{target}}, q_i$ output : $\Delta q_i, {}^{i-1}T_n, {}^0T_i$
At each time step : Compute : ${}^{i-1}T_i$ and $\frac{d{}^{i-1}T_i}{dq_i}$ Compute : $\frac{dP_n}{dq_i}$ and $\frac{d\varepsilon}{dq_i}$ Update : ${}^0T_i = {}^0T_{i-1} \cdot {}^{i-1}T_i$ Update : ${}^{i-1}T_n = {}^{i-1}T_i \cdot {}^i T_n$ Update : $\Delta q_i = -\alpha_i \cdot \frac{d\alpha\varepsilon}{dq_i}$

 Figure 8. Algorithm of the agent A_i

4.3 Influence of the parameters α_i

The behavior of each agent is dependant on its α_i coefficient. To explain the influence of these coefficient, let's take as example the 3R planar robot described on figure 9. Such robot is simple to compute, its behavior is easy to understand and it evolves in two dimensions. It's a redundant robot, *i.e.* several configurations may exist for a given position of the end effector (Note that, in the case of a 3R robot, it is no longer true for position and orientation). The figure 10 shows the motion of the robot for different settings of the parameters α_i . In the first example (left figure), the agent A_3 (closer to the end effector) had the highest value, the second one A_2 had a medium value, and the first one A_1 , the smallest; the robot mainly uses its last link to reach the target. In the second example (central figure), the agents A_1 and A_2 are set with the same coefficient and in the last example (right figure), each agent is set with the same parameters. The global trajectory is far from the optimal solution: the robot moves its end-effector away from the target to come back closer. As shows figure 10, the global behavior of the robot is highly dependant on the parameters.

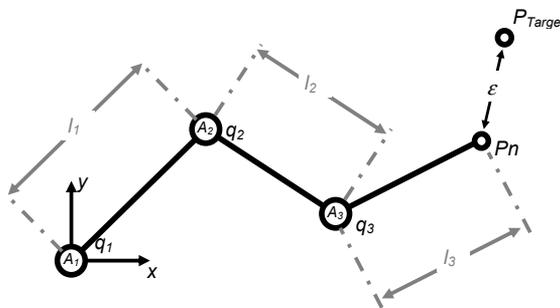


Figure 9. 3R planar robot

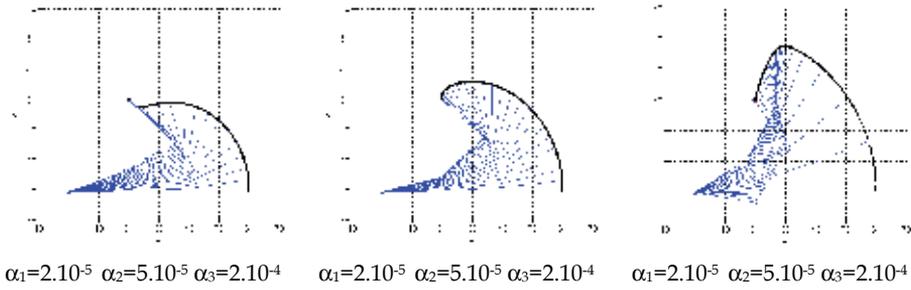


Figure 10. Example of trajectories for several sets of parameters

4.4 Stability

In order to apply the algorithm to a humanoid robot, the control of the static stability has been added. This is based on the same principal that for target tracking, rather than minimizing the distance between the hand and the target, the goal is to minimize the distance between the projection on the floor of the center of mass (COM) and the center of the footprint. To deal with the two objectives at the same time, the formula presented on the equation 6 is used.

$$\Delta q_i = -\gamma \cdot \alpha_i^{Target} \cdot \frac{d\mathcal{E}_{Target}}{dq_i} - (1-\gamma) \cdot \alpha_i^{COM} \cdot \frac{d\mathcal{E}_{COM}}{dq_i} \quad (6)$$

Where :

$$\gamma = \frac{D_2}{D_1 + D_2}$$

\mathcal{E}_{COM} is the distance between the projection of the center of mass and the center of the footprint

D_2 is the distance between the projection of the center of mass and the center of the footprint

D_1 is the distance between the projection of the center of mass and the closest edge of the footprint

These equations ensure that when the robot is perfectly stable ($\gamma=1$) the algorithm will control only target tracking. On the other hand, when the projection of the center of mass is on the edge of the footprint (ie. The robot is at the limit of stability and $\gamma=0$) hundred percent of the command is dedicated to stability. Between these two extremes, the ratio is linear. This algorithm had been implemented on the model of HRP2. The first simulations have immediately shown the efficiency of the method and the emergence of behavior. For example, the robot dedicates its right arm only for stability because the right arm is not useful for target tracking. One actual drawback of this technique is that the designer has twice as many parameters to set. One set of parameters for target tracking, and one set of parameters for stability. Experiments have shown that the parameters can be set arbitrary, likely due to redundancy. But finding a set of parameters that provide an optimal and/or human-like motion is not trivial. This is the reason why evolutionary computation has been used to evolve these parameters.

4.5 Evolutionary computation of the parameters

Evolutionary algorithm has been chosen for the following reasons: this algorithm doesn't need a model of the system, it provides several good solutions, it can optimize any kind of criteria and it can extract from local minima.

Chromosomal encoding: our goal is to optimize the parameters α_i^{Target} and α_i^{COM} . These parameters are directly encoded in the chromosomal string. The chromosomal string of an individual is composed of 44 float numbers in the case of HRP2: 22 for the target tracking objective, plus 22 for the stability. Twenty two is the number of joints minus six for the right leg (in order to ensure the feet are not moving on the floor; the right and left legs have been linked, i.e. each joint of the right leg is performing the same motion than the left leg joints) and two for the head. (the head joints are not considered here; we assumed that on the real robots these joints must stay available for the motion of the embedded vision system).

α_1^{Target}	α_2^{Target}	...	α_{22}^{Target}	α_1^{COM}	α_2^{COM}	...	α_{22}^{COM}
---------------------	---------------------	-----	------------------------	------------------	------------------	-----	---------------------

Table 4. Description of a chromosomal string

Initialization: at the beginning of the experiment, the various chromosomal strings are randomly filled in order to increase the diversity of the population. The range of the parameter is [-1 ; +1]. There is no need to extend this range; the important fact is the difference between the parameters, not their values. For example, by setting all the parameters to 1 the robot will have exactly the same behavior than by setting all the parameters to 0.1. The only difference is the speed of the motion. By weighing all the parameters with a coefficient K_s , it becomes possible to control the speed of the motion. Having negative values is important to push the robot back from unstable positions.

Fitness: the objective of the presented experiment is to have a human-like behavior in order to make the robot friendly with the people using it. Such behavior is not easy to transpose into equations which is why three fitness functions are considered: minimizing times during initial and final position of the hand, minimizing the traveled distance of the hand, and minimizing the energy according to equation 7. The behavior of the robot for each fitness function is discussed in the next section. For each individual, the fitness function was initially evaluated on 7 randomly selected targets. The obtained behaviors can't be generalized at any point of the workspace. The number of targets has been extended to 18 points strategically positioned into the workspace as shown on figure 11.

$$F = \sum_{t=0}^{t=T} \sum_{i=1}^n \frac{1}{2} m_i \cdot v_i^2(t) \quad (7)$$

Where :

F is the fitness function

T is the duration of the motion in time steps

n is the number of body composing the robot

m_i is the mass of the body i

$v_i(t)$ is the speed of the body i at time step t

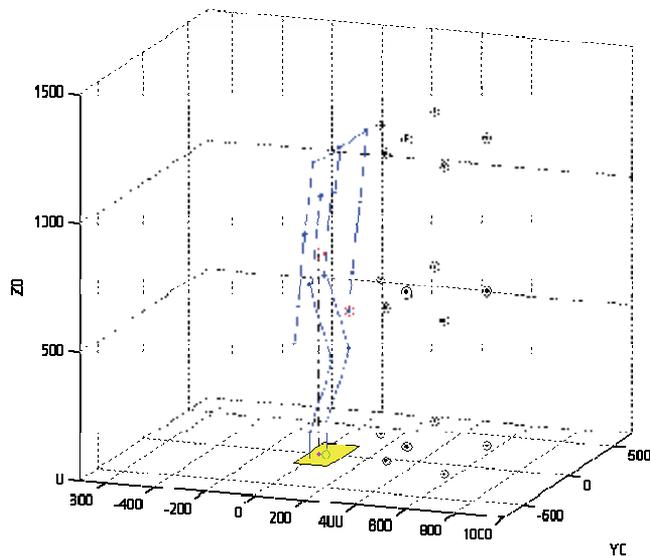


Figure 11. Position of the 18 targets during the evolutionary process

Crossovers: at the end of the evaluation of the population, the new generation is automatically generated. For each new individual, two chromosomal strings from the previous generation are selected using the roulette-wheel reproduction as described in (Nolfi & Floreano 2000). The new chromosomal string is generated by a random selection of the coefficients. For each new coefficient a random selection is made between the two same values in the two “parents”. This strategy is based on the principle of uniform crossovers (Mitchell, 1997).

Mutations: after the crossover process, ten percent of the population is randomly selected to mutate. Such percentage has been arbitrary chosen to ensure the equilibrium between a good exploration of the search space and the preservation of the best known individuals. A mutation consists of the selection of a coefficient α_i^x where i and x are randomly chosen. This coefficient α_i^x is then replaced by a new one, randomly selected in the range $[-1; 1]$.

4.6 Experimental results

Experiments have been conducted by simulation using the geometric model of the robot HRP2, including the center of mass of each body allowing the computation of the global center of gravity of the whole robot. Each joint is bounded in the same interval as on the real robot. The simulator is of course based on a centralized processor that simulates at each time step the behavior of each agent. This allows the simulation of the distributed system. The figure 12 shows the evolution of the fitness function in the first experiment where the purpose is to minimize the necessary time to reach the target from the initial position. Similar graphics have been produced for the other fitness functions. In the three cases, a convergence is observed that confirms the relevance of the chosen parameters.

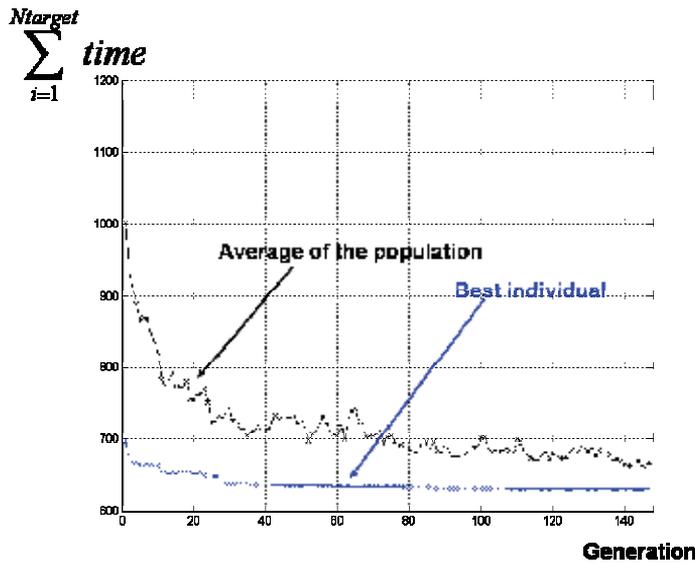


Figure 12. Evolution of the fitness function (time) for the whole population (average) and for the best individual of each generation

Time and distance: the fitness functions minimizing time and distance produce similar behavior on the robot. In both cases, the robot is able to reach the targets in any configuration. As the robot has been programmed to reach targets with the left hand, the right one has no influence in the target tracking. The coefficients have only been optimized by the evolutionary process to keep the stability of the robot. For example on figure 13, when the robot leans forward it puts back its right arm to compensate the move of the center of gravity. However, depending on the position of the target, the general motion of the robot is sometimes far from natural or human-like behavior. These observations motivated the implementation of a new fitness function consisting of minimizing the energy.

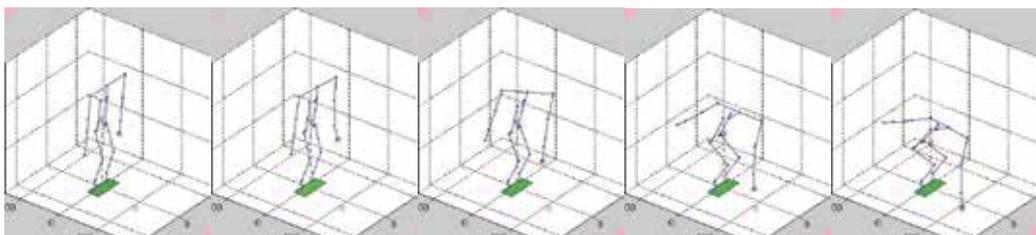


Figure 13. Example of motion of the robot

Energy: minimizing energy provides interesting results. First of all, the targets are still reached at any position of the workspace. The behavior of the robot is very smooth and natural, except for the targets over the shoulders. The robot moves its torso backward and then moves forward. This behavior is not really natural, because a human has the faculty of anticipating the end of the motion. However, in the present case a human would probably move one step backward in order to hold out its arm. Such behavior is not possible here because of the hypothesis that feet are linked to the floor.

Failures: in order to test the robustness of the system failures have been simulated. The first test was a failure on an actuator. As the agents are independent, the system continues working; the other agents naturally compensate the inactivity on a joint. The second test simulated failures on several joints. When too many actuators are inactive, the system is not always able to reach the target, sometimes even if it's always theoretically possible. These experiments have shown that the presented architecture is mainly intended for redundant robots. An observed conclusion is that the more the system is redundant, the more it can extract from local minima, mainly due to the fact that minima are not concerning the whole population. When an agent is attracted to a minima, the others, when moving, drag him from it, in the same way that a failure on an agent is compensated by the other agents.

Computation time: measurement of the computation time is now addressed. Note that the measurement has been made under Matlab, so we may expect better results with a compiled software. The computation for the whole robot requires 0.68ms per cycle. Half of this time is needed to compute the kinematic model, and about 44% for the computation of the derivative, 3% for the computation of the distances and 3% is needed to apply the command on the actuators. On a real distributed architecture, these delays may be divided by the number of agents.

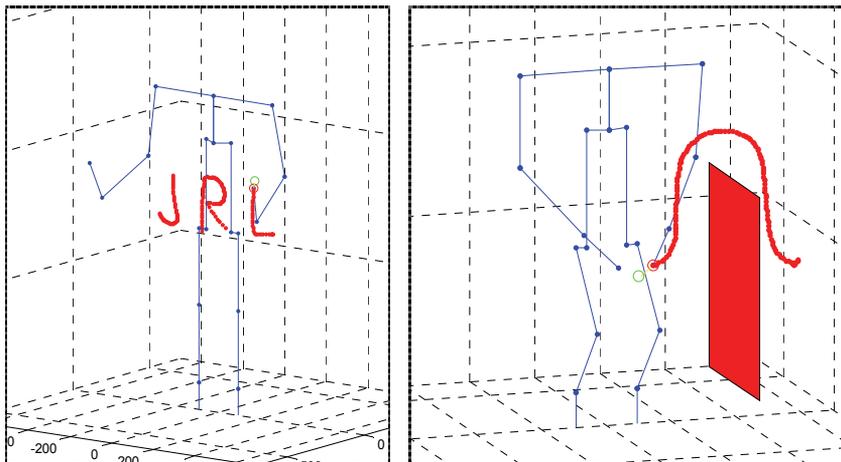


Figure 14. Example of trajectory followed by the hand of the robot (left), and simulation of obstacle avoidance (right).

Trajectory: improvements have been performed on the agent's behavior in order to perform trajectory following. The final goal of these improvements is to perform obstacle avoidance with external obstacles. The designed architecture is not described here, but the experiment has shown that following a trajectory with the end effector is possible as illustrated on figure 14. Controlling the speed of the end effectors is also possible, but the speed controller is currently centralized and the advantages of the distributed system are lost.

The results have been partially confirmed by the implementation of the algorithm on the OpenHRP platform where the dynamic model of the robot is fully implemented. Unfortunately, as the internal collision detection is not yet operational, an implementation on the real robot is currently too risky.

5. Conclusion and future works

This chapter deals with the combination between multi-agent systems and evolutionary computation. The first section describes an experiment of the evolutionary learning of an autonomous obstacle avoidance behavior. This first experiment proves the possibility of distributing a genetic algorithm into a real robot population. In the proposed architecture, new crossovers and mutation techniques have been proposed allowing the distribution of the algorithm.

The second part of the chapter deals with the decomposition of a unique robot into a multi-agent system. In the distributed proposed architecture, the behavior of the robot is dependant from a set of coefficients influencing the motion of each agent. These coefficients are evolutionary optimized. Several fitness functions have been experimented: time, distance and energy. Simulation results show that minimizing the energy is the best strategy, the system may be fault tolerant and the computation of the algorithm is very fast. Future works will be oriented on a survey on the behavior of the robot around singular configurations and the implementation of a collision avoidance module, preventing the robot from hurting himself. So as to keep the advantages of the presented architecture, a nice strategy would be to distribute the obstacle avoidance module. This implementation looks like being really challenging, because it will highly increase the communication between the agents. Once the collision avoidance is functional, the implementation on a real robot will be planned.

6. Acknowledgements

This work has been supported in part by the Robotics, Microelectronics and Computer Science of Montpellier (LIRMM, Montpellier, France), the Joint Robotics Laboratory (JRL-AIST, Tsukuba, Japan) and by a Japanese Society for the Promotion of Science (JSPS) grant.

7. References

- Arkin R.C. (1992). Cooperation without communication: multi-agent schema-based robot navigation, *Journal of Robotic Systems*, Vol 9, No.3, (April 1992), pp 351-364.
- Brooks R.A. (1986). A robust layered control system for a mobile robot, *IEEE Transaction on Robotics and Automation*, Vol. 2, pp.14-23.
- Drogoul A. & Ferber J. (1992). From Tom Thumb to the Dockers: some experiments with foraging robots, *Proceedings of the 2nd International Conference on Simulation of Adaptive Behavior*, pp. 451-459, Honolulu, 1992.
- Ferber J. (1999). *Multi-agent systems. An introduction to distributed artificial intelligence*, Addison Wesley, London
- Floreano D. & Mondada F. (1994). Automatic creation of an autonomous agent: genetic evolution of a neural-network driven robot. *From Animals to Animats 3: proceedings of Third Conference on Simulation of Adaptive Behavior*. Cambridge, MA: MIT Press/Bradford Books.
- Goldberg D.E. (1989). *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, reading, Mass, 1989.

- Lucidarme P. (2004). Anevolutionary algorithm for multi-robot unsupervised learning, *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 2210-2215, Oregon, Portland.
- Lucidarme P. & Simonin O. (2006). Le robot mobile Type 1. *Journée des démonstrateurs en automatique*, Angers, France
- Mitchell. T. (1997). *Machine learning*, McGraw Hill, ISBN 0070428077.
- Murase Y. Yasukawa Y. Sakai K. Ueki M. (2001). Design of a Compact Humanoid Robot as a Platform. *Nippon Robotto Gakkai Gakujutsu Koenkai Yokoshu*, Vol 19, pp. 789-790.
- Nolfi S. and Floreano D. (2000), *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-organizing Machines*, Bradford book, MIT Press, Cambridge, Massachusetts.
- Tucker B. & Arkin R.C. (1994), Communication in reactive multiagent robotic systems. *Autonomous robots*, Volume 1, No.1, pp. 27-52.

Embodiment of Legged Robots Emerged in Evolutionary Design: Pseudo Passive Dynamic Walkers

Kojiro Matsushita and Hiroshi Yokoi
Department of Precision Engineering, University of Tokyo
Japan

1. Introduction

Many legged robots have been developed, and some of them have already achieved dynamically stable bipedal and quadruped locomotion. Common characteristics of these conventional legged robots are that their motions are precisely manipulated by their control (i.e., gait and balance). In the case of ASIMO (Hirose et al., 2001) and HRP2 (Kaneko et al., 2004), their dynamically stable bipedal locomotion is based primarily on the Zero Moment Point (ZMP) concept, which was originally developed by Vukobratovic (Vukobratovic, 1969): the ZMP is a fictitious point on the ground plane, where the torques around the (horizontal) x and y axes (generated by ground reaction forces, inertial forces, and torques) are equal to zero. These robots dynamically stabilize their balance by manipulating the ZMP to remain within the support polygon area defined by their square feet, and their gait control is applied on top of the balance control. As a result, these conventional legged robots require rapid information processing and high-power drives to achieve locomotion. Therefore, they are characterized as having more complex control systems and larger energy consumption than those of biological locomotion.

On the other hand, these design requirements (i.e., fast information processing and high-power drives) are regarded as disadvantageous in biologically inspired robotics. In the field, the specific behaviors and/or structures of biological organisms are imitated (Vogel, 1998) (Alexander, 2002) with robotic technology. As a result, the robot systems have simple design requirements compared to conventional robot systems, and achieve complex tasks. Thus, the design effort tends to focus on control complexity and energy requirements. A representative instance is the Passive Dynamic Walker (PDW), which was originally developed by McGeer (McGeer, 1990). The PDW has no controller (i.e., no sensor and no motor), and the structure is based on the physical characteristics of human walking: passive hip joints, latch knee joints, and curved feet. The PDW walks down a slope, and its structure exploits gravity as the driving force. This design principle has been applied to the Cornell Biped (Collins & Ruina, 2005) and Denise at TU Delft (Wisse, 2003), and the powered passive dynamic walkers achieved passive dynamic walking on a flat plane.

In a summary of these two approaches, conventional robots require expensive design components to achieve dynamically stable locomotion. On the other hand, biologically inspired robots require physical characteristics that exploit their own dynamics to achieve

dynamically stable locomotion. After all, biologically inspired robots implicitly realize higher adaptability to specific tasks and environments (i.e., more distance traveled, less control complexity, and smaller energy consumption on a flat plane) than conventional robots. It is obvious, then, that physical characteristics greatly contribute to high adaptability.

In the field of embodied cognitive science, such physical characteristics are regarded as "embodiment" (Gibson, 1979). Embodiment is defined as special features in a body that result in high adaptability to tasks and environments. There is increasing evidence that embodiment enhances energy efficiency and reduces the complexity of control architecture in robot design (Brooks, 1999) (Pfeifer & Scheier, 1999). However, embodiment has only been demonstrated with heuristically developed robots, and the design process has not been revealed. One current agreement in embodied artificial intelligence hypothesizes that embodiment can emerge in robot design with the following biologically inspired reproductive process: (1) morphologies and controllers of robots are built in the physical world; (2) robots need to interact with physical environments to achieve a specific task; (3) robot settings are evaluated according to their task achievements, and the better ones are reproduced; (4) steps (2) to (3) are repeated (i.e., physical characteristics resulting in better task achievement tend to remain in the process); (5) specific features are hypothesized to form in the body (embodiment). At this point, such a reproduction process has already been implemented in evolutionary robotics, and the evolutionary reproduction process demonstrated a variety of locomotive robots (e.g., mainly crawlers) in the three-dimensional virtual world (Sims, 1994). However, this process has just shown the qualitative characteristics of embodiment, and no physical and numerical evidence of embodiment has been presented.

Therefore, in this paper, the focus is primarily on the physical and numerical illustration of the embodiment of legged locomotion. For this method, an evolutionary design system is implemented to generate various physical characteristics. The physical characteristics that reduce control complexity and energy consumption - embodiment - are then quantitatively investigated. Further objectives are to present a physical representation of the embodiment of legged locomotion and to demonstrate the use of robots on such a basis.

2. Evolutionary Design of Legged Robots

An evolutionary design system is proposed for emergence of embodiment on legged locomotion. The evolutionary design system consists of two parts. The first part is coupled evolution part, in which a genetic algorithm searches both morphology and controller space to achieve legged locomotion using a virtual robot in a three dimensional physics simulation. The second part involves evaluation of the evolved robots due to specifying their adaptability to tasks. All of the experimental parameters such as the simulation environment, the morphology and controller parameters, and the genetic algorithm are described in this section.

2.1 Three Dimensional Physics World

The design system is implemented using Open Dynamics Engine (ODE) (Smith, 2000), which is an open-source physics engine library for the three dimensional simulation of rigid body dynamics. The ODE is commonly used by program developers to simulate the dynamics of vehicles and robots because it is easier and more robust for implementing

joints, contact with friction and built-in collision detection than solving physical equations using the Euler method.

The environment configuration of the design system is given as sampling time 0.01 [sec], gravity 9.8 [m/s²] as gravity, friction 1.0, ground spring coefficient 5000N/m, ground damper coefficient 3000Ns/m.

2.2 Genetic Algorithm

The coupled evolution part is based on the general GA process, which starts with random genes and conducts 100 to 300 generations using a population size of 100 to 200 for each run. After all generations, the evolutionary process is terminated, and the next evolutionary process starts with new random genes. Such an evolutionary process is called seed. Table 1 lists setting values for the GA.

Parameter	Setting Value	Parameter	Setting Value
Seed	30 to 100	Number of Gene Locus	50 to 100
Generation	150 to 300	Crossover	5 to 10 %
Population	100 to 200	Mutation	5 to 10 %

Table 1. Setting Values in the GA

(i) Selection / Elimination Strategy

The design system uses an elite strategy that preserves constant numbers of higher fitness in the selection/elimination process due to its local convergence. At each generation, each gene acquires a fitness value. At the end of each generation, the genes are sorted from highest to lowest fitness value. The genes in the top half of the fitness order are preserved, while the others are deleted. The preserved genes are duplicated, and the copies are placed in the slots of the deleted genes. The copied genes are crossed at 5-10% and mutated at 5-10%.

(ii) Terminational Condition

The evolutionary process has two major terminational conditions for emerging legged locomotion: (1) An individual is terminated if the height of the center of gravity drops 90% below the initial height, and the individual acquires -1.0 [m] as its fitness; (2) If the position of the foot does not move more than 0.005 [m], the individual is terminated and acquires -1.0 [m] as its fitness. The former is a necessary condition to prevent falling or crawling solutions. The latter is a necessary condition to achieve cyclic movement (preventing still movement).

(iii) Fitness

Fitness in the evolutionary process is defined as the distance traveled forward for a constant period, which should be sufficient to achieve cyclic movement and short enough to economize the computational power. Normally the period is 6 to 10 [sec].

2.3 Gene Structure

A fixed-length gene is applied to the gene structure in the design system. It is because each gene locus in a fixed-length gene easily inherits specific design parameters during the evolutionary process. Besides, it is easy to save, edit, or analyze those design parameters.

In the gene structure, morphological and control parameters are treated equally (Fig.1) for the evolutionary process so that each locus contains a value ranging from -1.00 to +1.00 at an interval of 0.01. Figure 4-7 shows locus IDs corresponding to the following design parameters: L, W, H, M0, M1, M2, M3, M4, k, c, amp, and cycle, and these parameters are used with conversion equations.

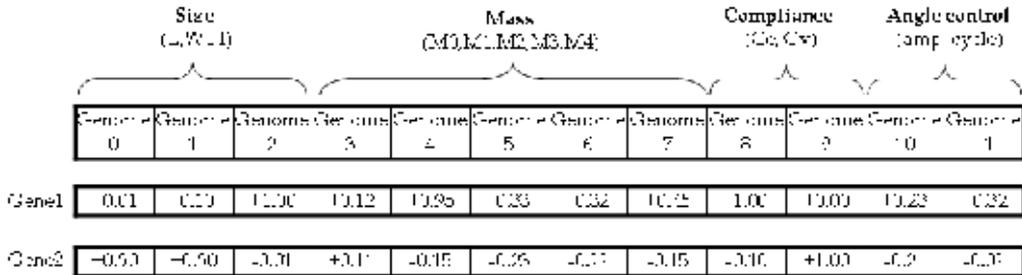


Figure 1. Concept figure of gene structure

2.4 Morphological Parameters

Morphology of a legged robot in the design system consists of five kinds of design components in Fig.2 and Table 2: joint type (compliant / actuated), joint axis vector, link size, link angle, and link mass. These physical components are viewed as basic components of a biological system (Vogel, 1999) and, therefore, it is hypothesized that the components satisfy presenting artificial legged locomotion.

		Link 1	Link 2
Size	Length [m]	0.1	0.1
	Width [m]	0.1	0.1
	Height [m]	0.1 to 0.5	0.1 to 0.5
Absolute Angle at Pitch (y) Axis [rad]		$-\pi / 3$ to $\pi / 3$	$-\pi / 3$ to $\pi / 3$
Mass [kg] (Total Mass X [kg])		X * 10-90%	X * 10-90%

Table 2. Basic link configuration

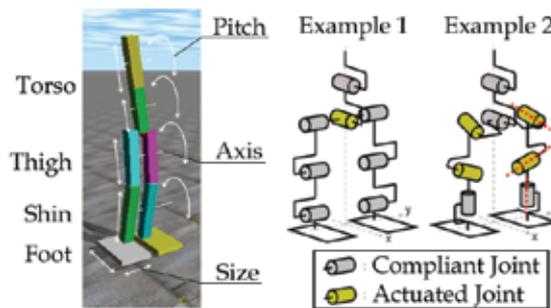


Figure 2. A basic representation of a physical structure

2.5 Control Parameters

It is hypothesized that a simple controller inevitably leads to the formation of special body features for stable legged locomotion in evolutionary processes. Simple rhythmic oscillators are applied in the design system due to identifying special features in a legged robot's body. Fig. 3 shows a basic representation of the joint structure. Contra-lateral set of joints are either rhythmic oscillators or compliance, which are determined in the evolutionary process. The characteristics of the oscillators are mainly determined by two types of parameters: amplitude and frequency (Table 3). In addition, all oscillators have the same wavelength, and contra-lateral oscillators are in anti-phase based on the physiological knowledge of gait control.

		Joint 1	Joint 2
Type	Compliance	Elasticity [N/m]	10 ⁻² to 10 ⁺⁴
		Viscosity [Ns/m]	10 ⁻² to 10 ⁺⁴
	Angle Control	Amplitude [rad]	0 to $\pi/2$
		Cycle [sec]	0.5 to 1.5
Axis Vector		X	-1.0 to +1.0
		Y	-1.0 to +1.0
		Z	-1.0 to +1.0

Table 3. Basic joint configuration

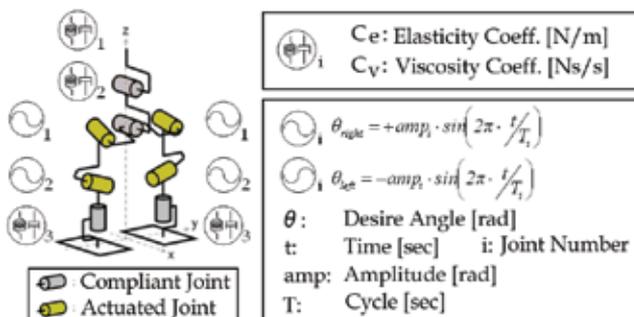


Figure 3. A basic representation of a control architecture

2.6 Evaluation Methods: Energy Consumption and Energy Efficiency

The design system targets legged robots, which achieve stable locomotion with less control complexity and smaller energy consumption than conventional legged robots. Therefore, energy consumption and energy efficiency are applied as the evaluation methods to qualify the evolved legged robots. The calculational procedure is described as follows.

In physics, mechanical work [Nm] represents the amount of energy transferred by a force, and it is calculated by multiplying the force by the distance or by multiplying the power [W] by the time [sec]. In the case of a motor, time and rotational distance are related with its angular speed, and the torque, which causes angular speed to increase, is regarded as mechanical work. Thus, power in rotational actuation is calculated with the following equation 1:

$$\text{Power [W]} = \text{torque [Nm]} * 2 \pi * \text{angular velocity [rad/s]} \quad (1)$$

Therefore, energy consumption for a walking cycle is represented with equation 2. Energy efficiency is computed as energy consumption per meter (equation 3). In this equation, total mass is ignored because it is set as a common characteristic.

$$\text{Energy Consumption for a walking cycle [J]} = \left(\sum_{i=0}^N \int_0^T |2\pi \cdot Tr_i(t) \cdot \dot{\theta}_i| dt \right) \cdot T \quad \dots(2)$$

$$\text{Energy Efficiency for Locomotion [J/m]} = \frac{\left(\sum_{i=0}^N \int_0^T |2\pi \cdot Tr_i(t) \cdot \dot{\theta}_i| dt \right) \cdot T}{Dis} \quad \dots(3)$$

N : The number of actuated joints

dt : Sampling time[sec] t : Time[sec]

$\dot{\theta}$: Angular Velocity[rad/s]

Tr : Torque at each joint T : A walking cycle[sec]

Dis : Distance traveled for a walking cycle[m]

3. First Experiment

The evolutionary design of biped robots is conducted to verify emergence of embodiment. In particular, focus on the relations between the physical configurations and the walking characteristics of the acquired biped robots, it is attempted to numerically reveal embodiment of the legged robots.

3.1 Morphological and Control Configuration for Biped Robots

Biped robots are constructed using nine rigid links: an upper torso, a lower torso, a hip, two upper legs, two lower legs, and two feet. These body parts are respectively connected at torso, upper hip, lower hip, knee, and ankle joints, and the robots have eight degrees of freedom.

			Torso	Hip	Knee	Ankle
Type	Compliance	Elasticity Coeff. [N/m]	10 ⁻² to 10 ⁺⁴			
		Viscosity Coeff. [Ns/m]	10 ⁻² to 10 ⁺⁴			
	Actuation (Angle control)	Amplitude [rad]	0 to $\pi/2$	0 to $\pi/2$	0 to $\pi/2$	0 to $\pi/2$
		Cycle [sec]	0.5 to 1.5	0.5 to 1.5	0.5 to 1.5	0.5 to 1.5
Axis vector	X	1	-1.0 to +1.0	-1.0 to +1.0	-1.0 to +1.0	
	Y	0	-1.0 to +1.0	-1.0 to +1.0	-1.0 to +1.0	
	Z	0	-1.0 to +1.0	-1.0 to +1.0	-1.0 to +1.0	

Table 4. Characteristic of joints (searching parameters colored in blue)

		Upper/ lower Torso	Hip	Thigh	Shin	Foot
Size	Length [m] (X axis)	0.1	-	0.1	0.1	0.1 to 0.5
	Width [m] (Y axis)	0.1	-	0.1	0.1	0.1 to 0.5
	Height [m] (Z axis)	0.1 to 0.5	-	0.1 to 0.5	0.1 to 0.5	0.05
	Radium [m]	-	0.05	-	-	-
Absolute angle at pitch (y) axis [rad]		$-\pi/3$ to $\pi/3$	-	$-\pi/3$ to $\pi/3$	$-\pi/3$ to $\pi/3$	$-\pi/3$ to $\pi/3$
Parallel displacement on y axis[m]		-	-	-	-	0 to 0.2
Total Mass 20 [kg] (a+2b+2c+2d=100%)		e	a	b	c	d

Table 5. Characteristic of links (searching parameters colored in blue)

Table 4 and table 5 lists control parameters (i.e., amplitude and frequency) and morphological parameters (i.e., size, weight, absolute angle of each link and selection of whether it is oscillatory or compliant, as well as its elasticity coefficient and viscosity coefficient if the joint is compliant or amplitude and frequency if the joint is an oscillator, and axis vector of each joint). In addition to this setting, joint settings are constrained to be contra-laterally symmetric around the xz plane as described in Section 2.5.

3.2 Results

The evolutionary design system performed thirty independent runs. At each time, the genetic algorithm started with a new random set of genomes (i.e., seed). Fig.4 shows fitness transitions of thirty seeds. We focus on the best genome from the nine most successful runs – the biped robots that locomote forward more than seven meters for ten seconds (Fig.5). Then, we analyzed the relationship between the morphologies and locomotion strategies of these robots.

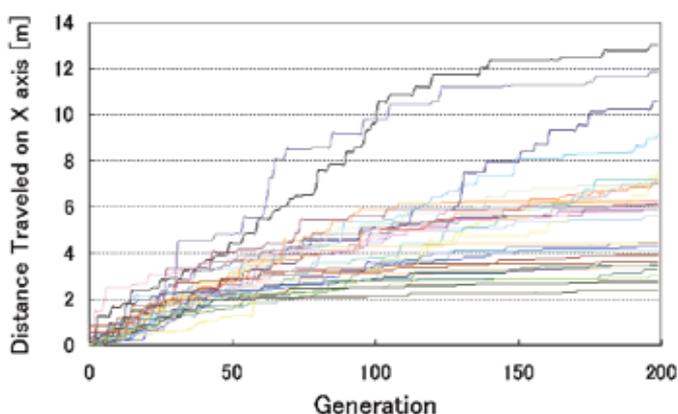


Figure 4. Transition of Best fitness (30seeds, 200generation, 200population)

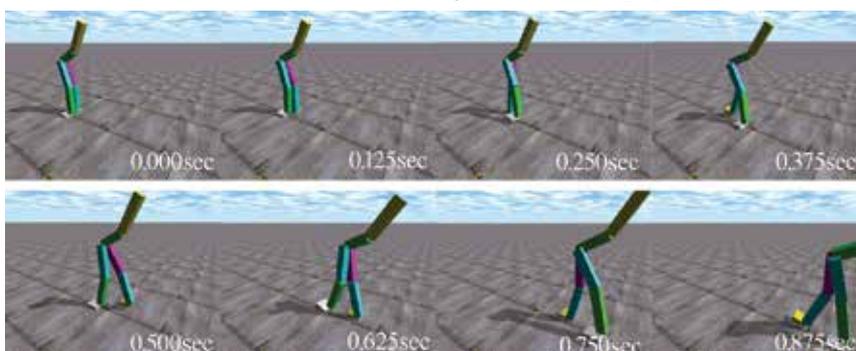


Figure 5. Walking scene of best fitness

Table 6 lists the performance of the best nine biped robots: the second column reports their distance traveled forward for 10 [sec]; the third column, their walking cycle; the fourth column, their angular velocity of oscillators; the fifth column, their energy efficiency; the sixth column, their numbers of contra-lateral set of actuated joints (i.e., four types - torso hip, knee, ankle joints). The biped robots indicating high energy efficiency tend to have less numbers of actuated joints in their system. It suggests that embodiment, which reduces

control complexity and energy consumption, is emerged in the system of the legged robots. Then, further analysis indicates that hip joints tend to become actuated joints, and knee joints tend to be compliant joints. Especially, focus on the characteristics of the compliant joints, they are categorized into three conditions: free joint, suspension joint, and fixed joint corresponding to the degree of elasticity and viscosity.

Seed	Distance [m]	Cycle [s]	Angular Velocity [rad/s]	Energy Efficiency [J/m]	Number of Actuated DOFs
09	13.0	1.02	0.50	6.0	2
11	7.3	1.05	0.32	7.0	1
02	7.7	1.17	0.38	7.7	2
00	10.6	1.05	0.55	8.2	3
14	7.0	1.01	0.45	10.0	2
22	11.9	1.06	0.99	13.1	3
08	9.2	1.04	0.81	13.8	3
21	7.1	1.08	0.69	15.2	3
17	7.2	1.04	0.88	19.1	3

Table 6. Performance of best 9 biped robots in order of energy efficiency (Energy efficiency is calculated with average torque 25.[Nm], and lower values indicate better performance)

	Upper hip joint	Lower hip joint	Knee joint	Ankle joint
Number	4	1	7	2

Table 7. Number of compliant joints among best 9 biped robots

Free Joint	Condition		Number of Types
	0 ≤ Ce < 10	0 ≤ Cv < 10	
Suspension Joint	10 < Ce ≤ 100	0 ≤ Cv < 10	5
Fixed Joint	Ce > 100	-	6
	-	Cv ≥ 10	

Table 8. Characteristics of compliant joints among best 9 biped robots (Ce: elasticity coefficient [N/m], Cv: viscosity coefficient [Ns/m])

3.3 Active control walker vs Compliant walker

In the previous section, it is confirmed that compliant joints have three conditions, however, it is not revealed that how the conditions contribute to the stable locomotion of the best nine legged robots. So, an additional experiment is conducted to verify roles of the compliant joints. The additional experiment proceeds as follows: (1) the evolutionary design system of biped robots conducts again under the condition, which compliance is not involved as design parameters; (2) the best biped robots in the design system – namely, active controlled walkers – are compared analyzed with the best biped robots in the previous design system – namely, compliant walkers. (The actively controlled walker indicates a biped robot without any compliant joint.)

As results of the additional experiment, Fig. 6 show joint angle trajectories of the compliant walker and the actively controlled walker, and Fig. 7 shows results of frequency analysis on the transitions. Here, the compliant walker has remarkable characteristics on hip and knee joint (as described in previous section) so that only those transitions are focused.

Among the varied behavior of the joints, it is observed that the knee oscillation in the compliant walker is induced by oscillators at other joints (self-regulation (Iida & Pfeifer, 2004)). Moreover, amplitude at 2 [Hz] in Fig.7(a) indicates ground impact absorption (self-stabilization) with compliance. That is, the appropriate state of compliant joints realizes these functions passively and dynamically during locomotion. Therefore, the robots which obtain these characteristics can be called pseudo-passive dynamic walkers. Moreover, these two functions serve as examples of the computational trade-off possible between morphology and controller, because compliant joints can be moved by energy input channels other than controlled motors and filter noise without computational power.

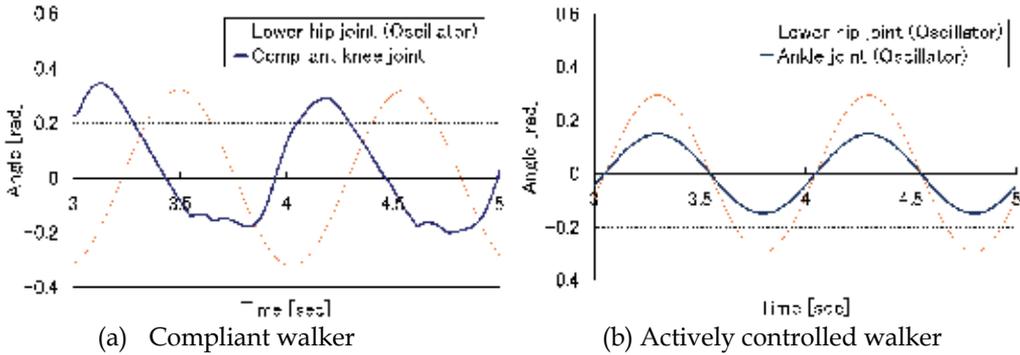


Figure 6. Joint angle trajectories of hip and knee joints

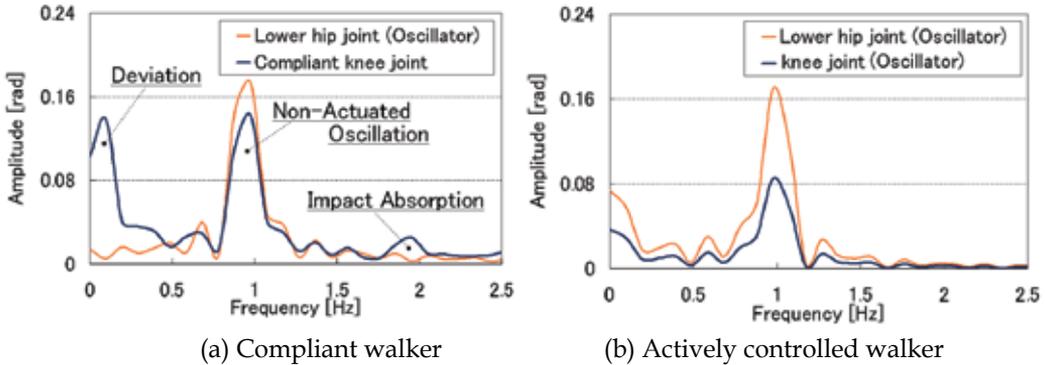


Figure 7. Frequency analysis (i.e., discrete Fourier transform) of joint angle trajectories of hip and knee joints

4. Second Experiment

The second evolutionary design is conducted for clarifying the embodiment: compliance. Basically, the setting parameters in Section 3.1 are applied to the evolutionary design and, for the purpose of narrowing its solution space to specify physical structures exploiting compliance, the condition, that restricts the numbers of actuated joints, is added to the system. Table 9 indicates joint configurations for the second evolutionary design, and a scheme for joint-type selection is as follows: one of four types of joint structures (i.e., either set of torso, hip, knee, and ankle becomes an actuated joint and other sets of the joints are compliant) is selected for a walker. The evolutionary design is conducted using 100 different random seeds, is run for 100 generations, and the population is comprised of 100 individuals.

			Torso	Hip	Knee	Ankle	
Type	Compliance	Elasticity Coeff. [N/m]	10 ⁻² to 10 ⁺⁴				
		Viscosity Coeff. [Ns/m]	10 ⁻² to 10 ⁺⁴				
	Actuation (Angle control)	Amplitude [rad]	0 to $\pi/2$	0 to $\pi/2$	0 to $\pi/2$	0 to $\pi/2$	
		Cycle [sec]	0.5 to 1.5	0.5 to 1.5	0.5 to 1.5	0.5 to 1.5	
	Selection of Joint Type	0 to 3	0	Act.	Comp.	Comp.	Comp.
			1	Comp.	Act.	Comp.	Comp.
			2	Comp.	Comp.	Act.	Comp.
3			Comp.	Comp.	Comp.	Act.	

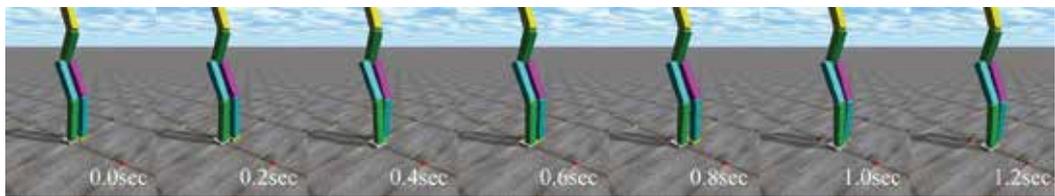
Table 9. Joint Configuration (searching parameters colored in blue)

4.1 Results: Walking Characteristics

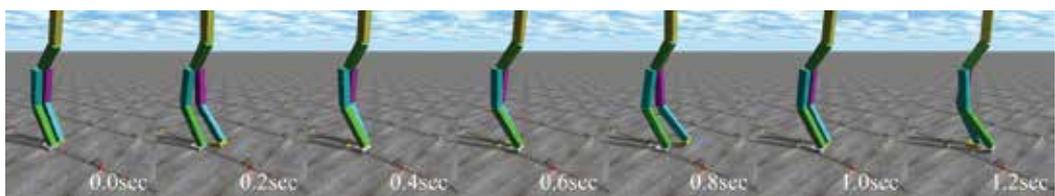
The evolutionary design generated six notable walks. In this section, their walking characteristics are described according to their joint structures.

(i) Hip actuated walkers

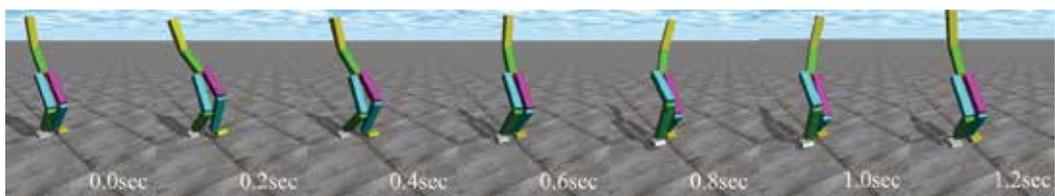
Hip actuated walkers are defined as walkers in which actuation is located at the hip, and the other joints are compliant. This pattern arose often (i.e., 55 out of 100 seeds) in the evolution process. The gaits can be characterized into three notable types: statically stable, dynamically unstable, and dynamically stable walks.



(a) Statically Stable Walk



(b) Dynamically Unstable Walk



(c) Dynamically Stable Walk

Figure 8. Representative hip actuated walkers

The statically stable walk is achieved most often among the hip actuated walkers (Fig.8a): they increase their mechanical stability and keep a narrow amplitude range in their oscillation so that their COG-Xs remain within their supporting polygon while walking. Fig.8 (b) shows a dynamically unstable walk. It walks with a tottering gait because the edges of its feet randomly contact the ground. So, its performance is unstable even on the flat plane. Meanwhile, Fig.8(c) shows a dynamically stable walk. The main feature is the axis vector of its hip joint: the oscillations at the hip axis synchronously move not only the legs in the sagittal plane but also the torso in the lateral plane. It is revealed that hip actuation enhances its stability with this physical feature.

Overall, the hip actuated walkers tend to exploit compliance only a little, and achieve their walks mainly by their actuation.

(ii) Knee actuated walker

Knee actuated walkers are defined as walkers in which actuation is located at the knee and all other joints are compliant, yet the genetic algorithm rarely generated (i.e., 3 out of 100 seeds) such walkers during evolution. The walkers commonly present a unique solution. Fig.9 shows the scene that the walker rotates the knee at the Z axis, and does not fall over for 6 seconds. That is, the walker finds a solution to elude the two termination criteria (i.e., (1) the height of the center of gravity drops 90% below the initial height; (2) the position of the foot does not move more than 0.005 [m]), and therefore remained.

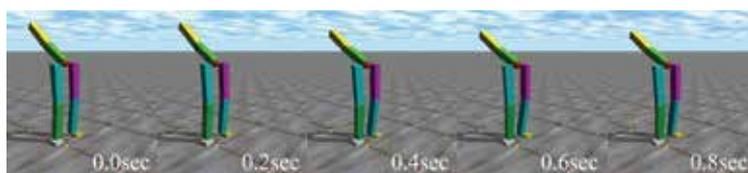


Figure 9. Representative knee actuated walker

(iii) Ankle actuated walker

Ankle actuated walkers are defined as walkers in which actuation is located at the ankle and all other joints are compliant, and also hardly generated solutions (i.e., 12 out of 100 seeds) during evolution. Its walk exploits compliance: the ankle is actuated and, then, the compliance in the walker synchronously moves the hip, knee, and torso joints by the actuation. Fig.10 shows the walking scene. It is observed that the physical structure of the walker is regarded as a laterally oscillating spring while walking. Thus, it indicates that compliance contributes to its stable walk.

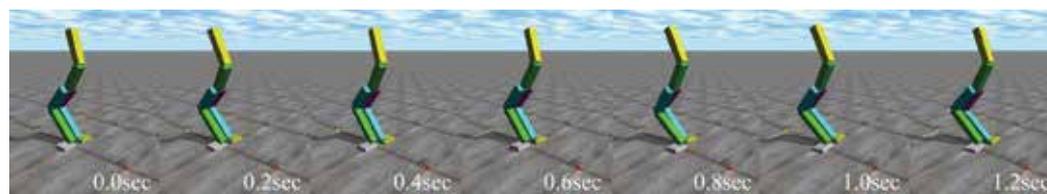


Figure 10. Representative ankle actuated walker

(iv) Torso actuated walkers

Torso actuated walkers are defined as walkers in which actuation is located at the torso and all other joints are compliant, and produced the second most common solutions (i.e., 29 out of 100 seeds) during evolution. Fig.11 shows the representative walking scene. The walker transfers a torso oscillation (actuation) in the lateral plane to hip oscillations (compliance) in the sagittal plane by exploiting its joint structure and material properties, and achieves stable walking.

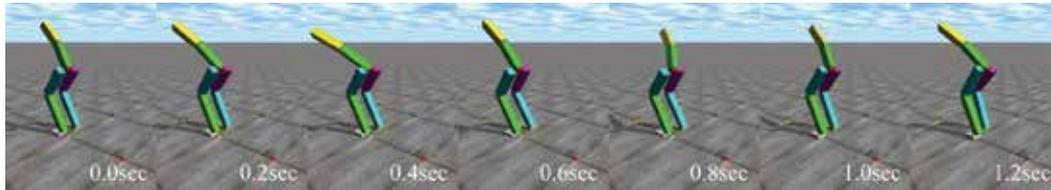


Figure 11. Representative torso actuated walker

4.2 Physical Representation of Embodiment

For illustrating the relations between physical structures, distances traveled, and energy consumption, the best fitness from 100 independent evolutionary runs are plotted on a two-dimensional graph (Fig.12: energy consumption on the vertical axis, distance traveled on the horizontal axis, and markers representing joint structures). It is characterized that each type of walkers is distributed around a certain area on the graph: the hip actuated walkers around the center; the knee actuated walkers around zero; the ankle actuated walkers around the left bottom; the torso actuated walkers around the bottom. Table 10 lists the best performance (i.e., their distances traveled, energy consumption, and energy efficiencies) in each type.

In terms of the rate of solutions generated, the evolutionary design generated the most hip actuated walkers. Meanwhile, the torso and ankle actuated walkers achieved higher energy efficiencies (energy consumption divided by distance traveled) so the rate does not relate to the emergence of embodiment.

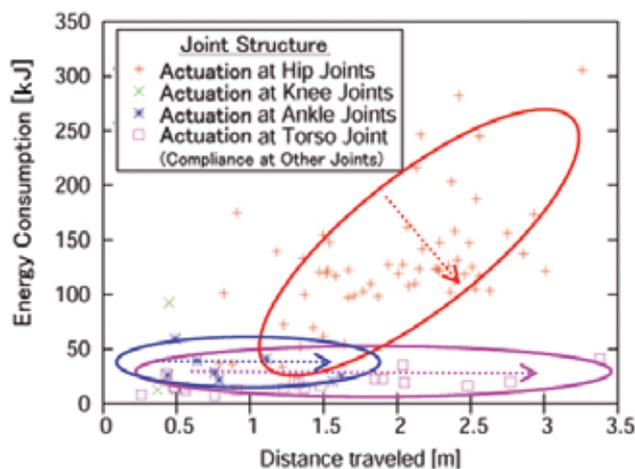


Figure 12. A physical representation of embodiment. It illustrates the relations between, joint structure, energy consumption, and distance travelled. Circles indicate distribution of four types of walkers, and arrows indicate the tendency of specific physical characteristics

For physical features, high fitness (i.e., distance traveled) of each walker tends to have specific physical features: (1) the ankle actuated walkers have high compliance at hip for the sagittal rotation, knee and torso for the lateral rotation; (2) the hip actuated walkers have low compliance at knee, ankle, and torso (i.e., only the hip is joints with mobility); (3) the torso actuated walkers have high compliance at hip for the sagittal rotation, low compliance at knee and ankle (i.e., the thigh and shin are regarded as one link). In particular, the walkers with high energy efficiency tend to characterize the specific physical features. Thus, Fig.12 indicates the joint structures and material properties (i.e., special physical features) and distance traveled and energy consumption energy efficiency (i.e., evaluation of embodiment). Then, the walkers with the special physical features have high evaluation on distance traveled and energy efficiency. That is, Fig.12 indicates a physical representation of the embodiment specifying pseudo passive dynamic walkers (the right bottom in the figure is best solutions).

Actuated Joint	Hip	Knee	Ankle	Torso
Number of Best fitness (Total 100 seeds)	56	3	12	29
Max Distance Traveled [m] (For 6 seconds)	3.04	0.48 (fall)	1.62	3.42
Energy Consumption [kJ] (For 6 seconds)	120.	-	25.	45.
Energy Efficiency [kJ/m]	39.4	-	15.4	13.1

Table 10. Best Performance of Four Types of Walkers

5. Development of Novel Pseudo Passive Dynamic Walkers

In this section, the physical representation of the embodiment (Fig.12) is utilized for development of pseudo passive dynamic walkers (PPDW): specific physical features are extracted from the representation, and simplified for developing novel pseudo passive dynamic walkers.

5.1 Novel Pseudo Passive Dynamic Walkers

The best embodiment in the representation (Fig.12) illustrates a structure, which is high compliance at the hip, low compliance at the knee and ankle, and actuation at the torso, and which achieves stable locomotion by transferring actuation power from lateral oscillation at the actuated torso to sagittal oscillations at the compliant hip. Then, such biped robots are simplified, and a novel biped PPDW is design as shown in Fig.13 (a). Fig.13 (b) shows walking mechanism of the Biped PPDW: it exploits its own physical features (i.e., actuation, gravitational, and inertial forces) for taking steps.

Moreover, an evolutionary design of quadruped robots is also conducted as the one of biped robots. Then, the design system also acquire embodiment for quadruped robots as shown in Fig. 13 (right). It basically represents the same design principle as the biped PPDW because the torso actuation transfers to the hip and shoulder joints to move. The conceptual walking mechanism is shown in Fig.13 (c).

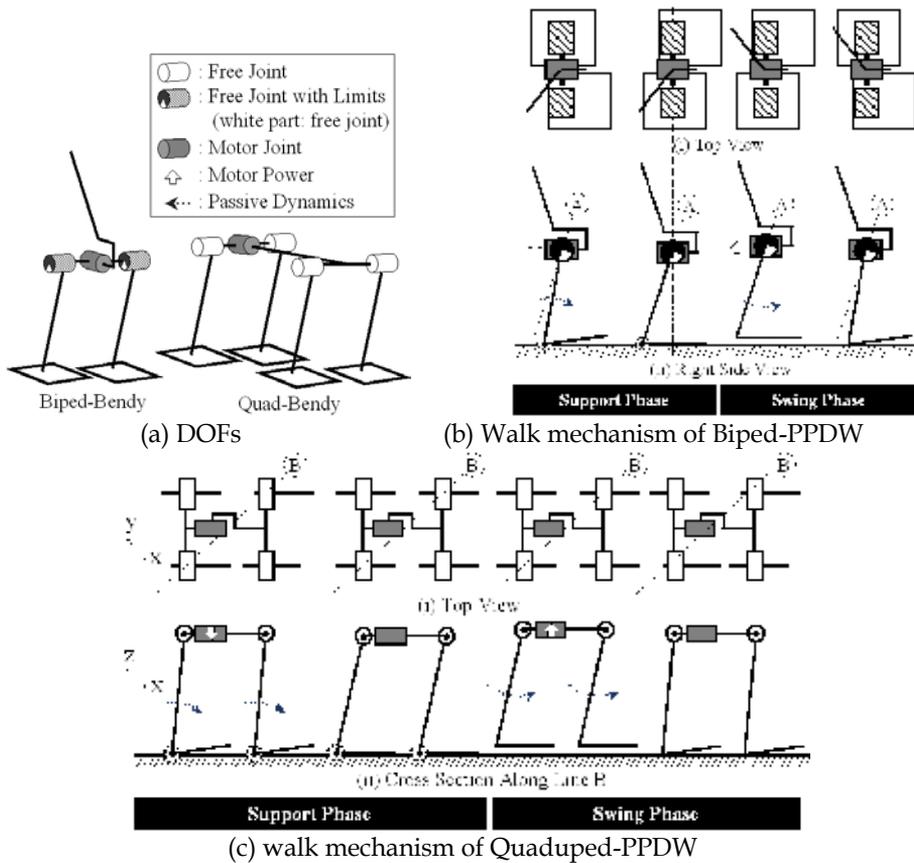
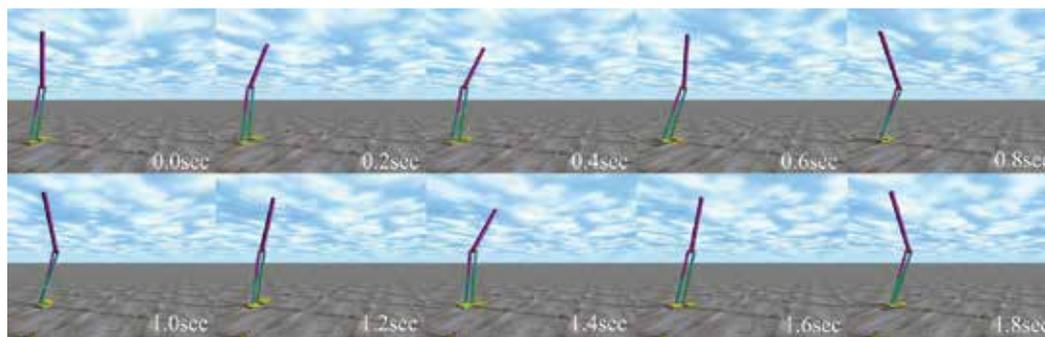


Figure 13. Biped and quadruped pseudo passive dynamic walkers (PPDW)

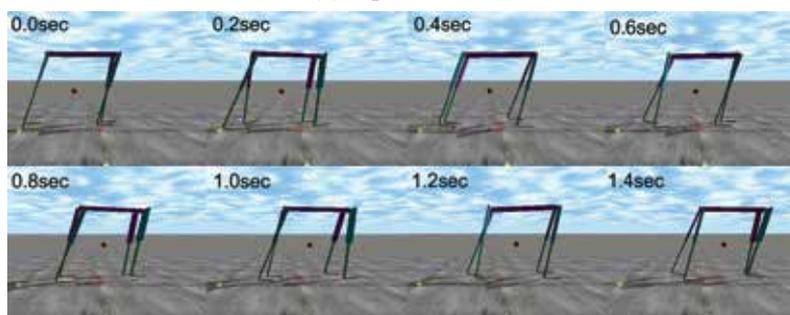
5.2 Demonstration in Real and Virtual World

The biped and quadruped PPDWs are developed for verification of their embodiments in both virtual and real world. The ODE is used for the implementation of virtual PPDWs. A robotic development kit is applied for the development of real PPDWs. The robotic development kit characterizes using plastic bottles as frames of robot structure, RC servomotors as actuated joints, and hot glues for connecting them (Matushtia et al, 2007). This unique approach has advantages of shorting machining and building time and enabling easy assembly and modification for beginners. It is not for developing precisely operated robots but the kit is durable enough to realize the desired behavior.

As results, the biped and quadruped PPDWs were developed in both virtual and real world, all the PPDWs achieved desire stable locomotion (Fig.14 and Fig.15). Although the real quadruped PPDW achieved similar performance to the virtual one, the real biped PPDW performed less than the virtual one (i.e., slower). It seems that the biped PPDW requires more dynamical stability than the quadruped PPDW so that it is difficult to tune the parameters of the biped PPDW for better performance. However, it is significant that even the real PPDWs, which are developed with the rough developmental kits, performed desire locomotion. Therefore, the embodiment is highly adaptive to low-cost and stable locomotion on flat plane.

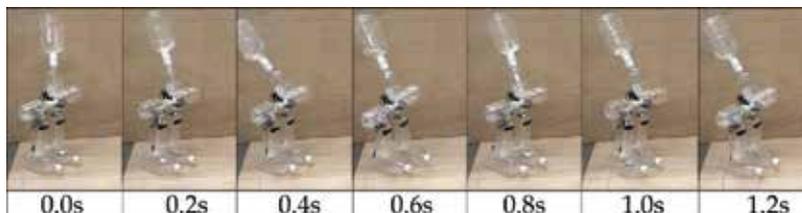


(a) Biped PPDW

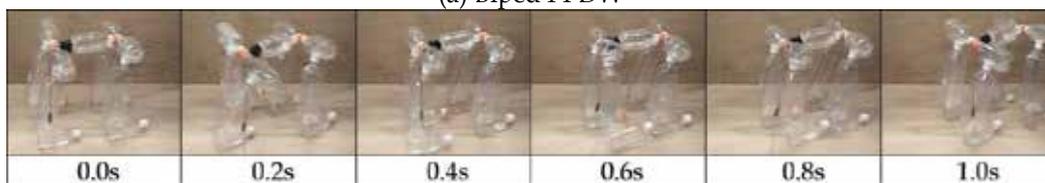


(b) Quadruped PPDW

Figure 14. Walk scenes of the PPDWs in virtual world



(a) Biped PPDW



(b) Quadruped PPDW

Figure 15. Walk scenes of the PPDWs in real world. The robots are developed with a robotic development kit for creative education. <<http://www.koj-m.sakura.ne.jp/edutainment/>>

6. Conclusion

An objective of this paper is to illustrate a physical representation of the embodiment on legged locomotion. Embodiment is here defined as physical features that reduce control complexity and energy consumption of legged robots. In this method, the embodiment of

biped robots is explored by the coupled evolution of morphology and a controller. As a result, (1) the first evolutionary design verified the emergence of embodiment: two functions of compliance contributed to dynamically stable locomotion; (2) the second evolutionary design specified the physical features (i.e., compliance and structures) and the effects: the biped robots resulting in higher energy efficiency tended to have specific physical features (i.e., a physical representation of embodiment). Eventually, the representation led to the development of novel pseudo-passive dynamic walkers, and those robots demonstrated legged locomotion with one motor in both the virtual and real worlds.

7. Acknowledgements

K. Matsushita is financially supported by Research Fellowships of Japan Society for the Promotion of Science for Young Scientists.

8. References

- Hirose, M.; Hirukawa, Y.; Takenaka, T. & Hirai, K. (2001). Development of Humanoid Robot ASIMO. *Proceedings of International Conference on Intelligent Robots and System*.
- Kaneko, K.; Kanehiro, F.; Kajita, S.; Hirukawa, H.; Kawasaki, T.; Hirata, M.; Akachi, K. & Isozumi, T. (2004). Humanoid robot HRP-2. *Proceedings of 2004 IEEE International Conference on Robotics and Automation*, pp. 1083- 1090
- Vukobratovic, M. & Stepanenko, J. Juricic. (1972). On the Stability of Anthropomorphic Systems. *Mathematical Bioscience*. Vol.15, pp.1-37.
- Taga, G.; Yamaguchi, Y. & Shimizu, H. (1991) Self-organized control of bipedal locomotion by neural oscillators in unpredictable environment. *Biological Cybernetics*, Vol.65, pp.147-159.
- Vogel, S. (1998). *Cats' Paws and Catapults*. W.W.Norton & Company.
- Alexander, R. (2002). *Principles of Animal Locomotion*. Princeton University Press.
- McGeer, T. (1990) Passive dynamic walking. *International Journal of Robotics Research*. Vol.9, No.2, pp.62-82.
- Collins, S. & Ruina, A. (2005) A bipedal walking robot with efficient and human-like gait. In *Proceedings of 2005 IEEE International Conference on Robotics and Automation*.
- Wisse, M. & Frankenhuyzen, I. (2003) Design and construction of MIKE: a 2D autonomous biped based on passive dynamic walking. *Proceedings of International Symposium on Adaptive Motion and Animals and Machines*.
- Gibson, J. (1979). *The Ecological Approach to Visual Perception*. Boston: Houghton-Mifflin.
- Brooks, R. (1999). *Cambrian intelligence: the early history of the new AI*. MIT Press, Cambridge, MA.
- Pfeifer, R. & Scheier, C. (1999). *Understanding Intelligence*. MIT Press.
- Sims, K. (1994) Evolving Virtual Creatures. *Computer Graphics Annual Conference Proceedings*, pp.43-50.
- Smith, R. (2000) Open Dynamic Engine. URL: <http://www.q12.org/>.
- Iida, F. & Pfeifer, R. (2004). Self-stabilization and behavioral diversity of embodied adaptive locomotion. *Embodied Artificial Intelligence*, LNCS3139, pp.119-129.
- Matsushita, M.; Yokoi, H. & Arai, T. (2007). Plastic-Bottle-Based Robots in Educational Robotics Courses -Understanding Embodied Artificial Intelligence-. *Journal of Robotics and Mechatronics*. Vol.19, No.2, pp. 212-222.

Action Selection and Obstacle Avoidance using Ultrasonic and Infrared Sensors

Fernando Montes-González, Daniel Flandes-Eusebio and Luis Pellegrin-Zazueta

*Departamento de Inteligencia Artificial, Universidad Veracruzana
México*

1. Introduction

The use of genetic algorithms in behavior based robotics is employed to adjust the parameters of the behavioral patterns that the robot executes. In order to model these patterns, Khepera robots can be used as one of many options. Therefore, in this paper we are proposing to expand the basic sensing capabilities of the robot by the development of a rapid ultrasonic detection turret for the Khepera II robot. The use of this turret favors the detection of different kinds of objects. Our purpose is to simulate the kind of behavior that a robot encounters in the navigation of human environments. For example, objects like short tables or desks standing on their legs above the floor. The standard Khepera II robot is equipped with eight infrared sensors, thus we designed an ultrasonic turret with three more sensors in order to grant the robot with detection capabilities for objects with short legs. Later on, we developed an obstacle avoidance behavior with the use of genetic algorithms. Therefore, in section 2 a brief background on genetic algorithms is provided. In section 3, we discuss some topics related to ultrasonic sensing, and we introduce the integration of these sensors into the Khepera. Section 4 describes the employed parameters for the use of the genetic algorithm and the robot equipped with ultrasonic and infrared sensing capabilities. The description of some experiments for testing the setup of the robot is explained in Section 5. Then, in section 6 we explore the integration of the extended capabilities of the Khepera within a foraging task with Action Selection. Finally, we provide a general conclusion in section 7.

2. Evolutionary Robotics and Genetics Algorithms

In writing several examples of solutions have been provided for the development of robot behavior. Commonly, the implementation of a particular behavior is carried out once the experimental setup is established. For example, robots can be set in a semi-structured environment where they solve particular tasks. Take the work of Bajaj and Ang Jr. for instance (Bajaj & Ang, 2000), where the *standard* Khepera has to solve a maze by avoiding obstacles and following walls. In the mentioned work and similar works, the use of Genetic Algorithms (Holland, 1975) is preferred over existent evolutionary methods like: Evolutionary Strategies, Genetic and Evolutionary Programming and Co-evolution. The

development of a basic genetic algorithm is a solid approach for starting to work on evolutionary robotics. Therefore, in our paper we chose the use of this method for tuning an obstacle avoidance behavior. It is important to notice that the resultant behavior, which is shaped and nearly optimized by the use of the genetic algorithm, ultimately depends on the right choice of the fitness function. Next we provide a brief background on genetic algorithms to support the development of our work.

The use of genetic algorithms and neural networks (Nolfi & Floreano, 2000) offers a good solution to the problem of modeling an obstacle-avoidance behavior in maze-like environments. Neural controllers require the setup of a chosen topology, and this can be done by the use of some rules of thumb. Once the topology is decided the weights of the neural controller have to be configured. A common approach relies on the use of backpropagation training that is a form of supervised learning where the neural net has to learn a known response to a particular configuration of the sensors in the robot. The general misclassification error is calculated and decreased over time when the neural network is trained. However, this kind of learning requires the design of training and validation data. On the other hand, the use of genetic algorithms is a form of gradient ascent approach that refines at each step of the optimization the quality of initial random solutions.

The optimization of neural controllers with genetic algorithms requires the representation, as a vector, of the weights of the neural controller. Then, a common practice consists of a direct encoding of the neural weights as an array that represents the genetic material to be manipulated by artificial evolution. A single neural controller represents one of the many individuals that form a population, which in turn are candidates for providing a good solution to the task that is to be solved. On the other hand, the fittest individuals of one population are used to breed the children that will be evaluated in the next generation. Therefore, the quality of a solution ('fitness') is measured to acknowledge whether a candidate solution is or not a good solution to the behavior we are trying to model. If the fitness of all candidate solutions is plotted, we will end up with a convoluted space where all possible fitness solutions can be represented. Therefore, this fitness landscape is formed by mountains and valleys, where landmarks in the mountains represent good quality solutions and landmarks near valleys are poor solutions.

The search of the best solution within a fitness landscape requires the guidance of the genetic algorithm to move uphill to find improved solutions. Nevertheless, a few downhill steps may be necessary in order to climb to the highest mountain. Therefore, exploration is guided by the use of a fitness formula that defines the behavior to be shaped, and three genetic operators are employed to create new solutions from existent ones. Thus, the current evaluated population spawns a new generation by the *selection* of a subset of the best individuals, the reproduction of the best individuals in pairs by the *crossover* of their genetic material, and the *mutation* of some of the material genetic of the individuals in the new population. The application of these operators to an initial random population of weights will produce refined solutions over time, and then the fitness evaluation will shape the final behavior through the breeding of the fittest individuals. However, few iterations are needed before this occurs.

3. Ultrasonic Sensing and its Integration with the Khepera Robot

The standard capabilities of the Khepera have already been extended in other works. However few are related to the use of ultrasonic sensors. Take for instance the work of

Chapman, et al. (Chapman, et al., 2000) where a wind-sensor is built for solving a maze. The work of Webb, et al. (Webb, et al., 2005) provided ears to a Khepera in order to simulate a female cricket. Furthermore, Odenbach, et al. (Odenbach, et al., 1999) fitted a wireless communication system on the Khepera to allow the identification of another Khepera in contrast to surrounding obstacles. Böndel, et al. (Böndel, et al., 1999) extends the Khepera to pick up small holed cubes. Another extension of the Khepera by Goerke, et al. (Goerke, et al., 1999) allows the robot to play golf. In contrast, the work of Winge (Winge, 2004) is the closest work to the one presented here. However, he makes use of the SRF04 sonar¹ and a major inconvenience is the programming of the Khepera microcontroller to estimate the measured distance to the objects.

In our work we are extending the sensing capabilities of the Khepera robot (Mondana, et al., 1993) with the use of a rapid ultrasonic detection turret; thus, we introduce how these sensors work. They use a non-audible pulse of 40 KHz, which travels through the air. When an object is close to the transmitter-receptor of the ultrasonic sensor, sound waves bounce back from that object and this bounce-back of the sound is then detected by the ultrasonic sensor. An elegant solution for the measurement of distance using ultrasonic sensors consists on the calculation of the time-estimation of the bounce-backed sound to the receptor. In order to calculate distance, we assume that the speed of the sound in the air is already known, and can be calculated using the rule of three from the next formula

$$D = v_s \cdot \frac{t_{propagation}}{2} \quad (1)$$

where: $v_s = 340[\text{m/s}]$ is the propagation speed of the acoustic waves in the air; and $t_{propagation}[\text{s}]$ is the total propagation delay of the acoustic waves.

However, by following an approach such as this requires the use of an analogue-digital converter to transform the resultant data into a digital format that can be processed by the Khepera's microcontroller. An alternative method calculates the distance from the measured intensity of the bounced-back sound into the receptor. As a result, we obtain an analogue signal that can be passed to the microcontroller of the Khepera. The latter method is less accurate, though is cheap and requires a reduced amount of electronic components. The measurement of the bounced-back sound is a rapid solution that makes a popular choice for prototype development. Thus, for the design of a rapid ultrasonic detection system we measure the intensity of the sound that bounces-back from objects in the near range.

The General I/O (*Gen I/O*) extension turret of the Khepera is widely used to expand the standard capabilities of this robot. The Gen I/O for these purposes offers 8 digital inputs; 2 analog inputs with adjustable gain; 1 analog differential input; 4 digital low power outputs; 1 digital high-power output; and 1 motor control (full H Bridge). Therefore, we will fit three ultrasonic sensors on the top of a Khepera robot in order to grant the robot with the capability of detecting farther short-legged objects than the infrared sensors can detect. The use of the Gen I/O permits the connection of two of the three ultrasonic sensors to the analog inputs with adjustable gain, and the third one to the analog differential input. The three sonar inputs are transformed by the analog convert of the robot into data that can be

¹ A modular sonar fitted with a transmitter and a receiver that employs input signal-conditioners, and the output of a digital pulse signal, to measure distance using external logic devices.

interpreted by the microprocessor of the Khepera. Then, for the construction of the ultrasonic turret, we should take into account that the distance calculated from detected obstacles should be converted into an analog value ranging from 0 to 5 Volts. The interpretation of this digital value is dependent of the application to be implemented. In our case, we scale this range to 0-1024 values for the readings of detected obstacles. Next, the ultrasonic turret is powered from the robot with a source of 5 volts with a maximum electrical current of 250 mA. This value of the electrical current is enough for the analog circuit on the extension board to work. The design of the transmitter, shown in two blocks, is presented in Figure 1(a).

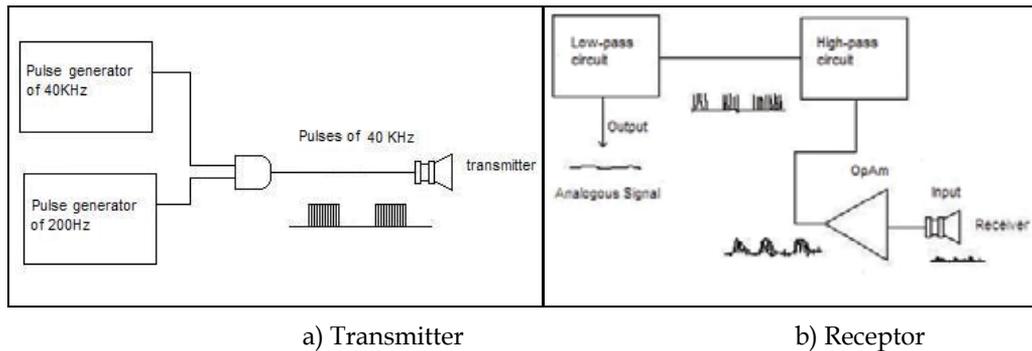


Figure 1. Block diagram of the ultrasonic transmitter/receptor

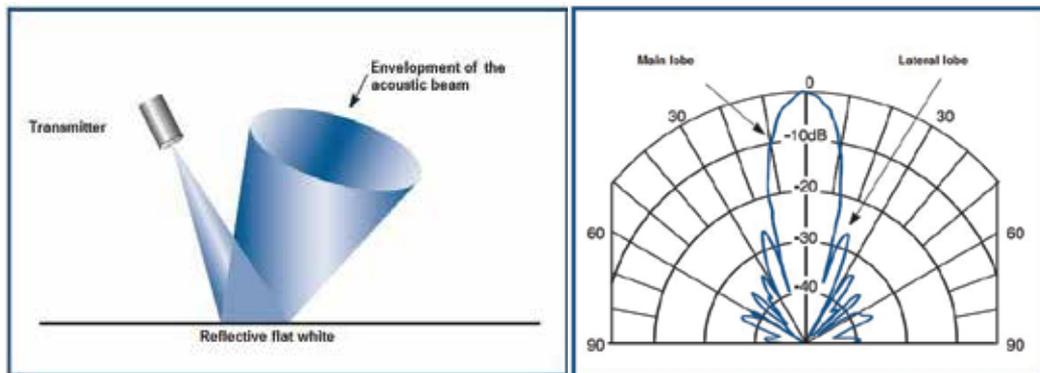


Figure 2. Behavior of the acoustic beam

The transmitter works in the follow manner, a pair of pulse generators are employed, one generator outputs pulses of 40 KHz and the second produces a 200 Hz signal. Then, both generators produce a train of pulses. The pulse train uses a carrier signal of 40 KHz. The implementation of the ultrasonic transmitter makes use of regular oscillator circuits like the LM555, and a couple of standard digital gates to generate the pulse train. Next, standard inverse-gates reduce the width of the pulses and the signal is then feed into the ultrasonic transmitter. The receptor circuit is presented in Figure 1(b), and for its implementation we use an OpAm (Operational Amplifier) such as the LM324 that facilitates the use of non-symmetrical sources (*GND and +5V*).

The transmitter is able to reach objects within a scope defined by a 10 degree angle scope (Fig. 2(a)); thus allowing a degradation of 3dB when the object is out of range (Fig. 2(b)).

This degradation provides a good estimation in accuracy of the measured distance of objects within sight and out of range.

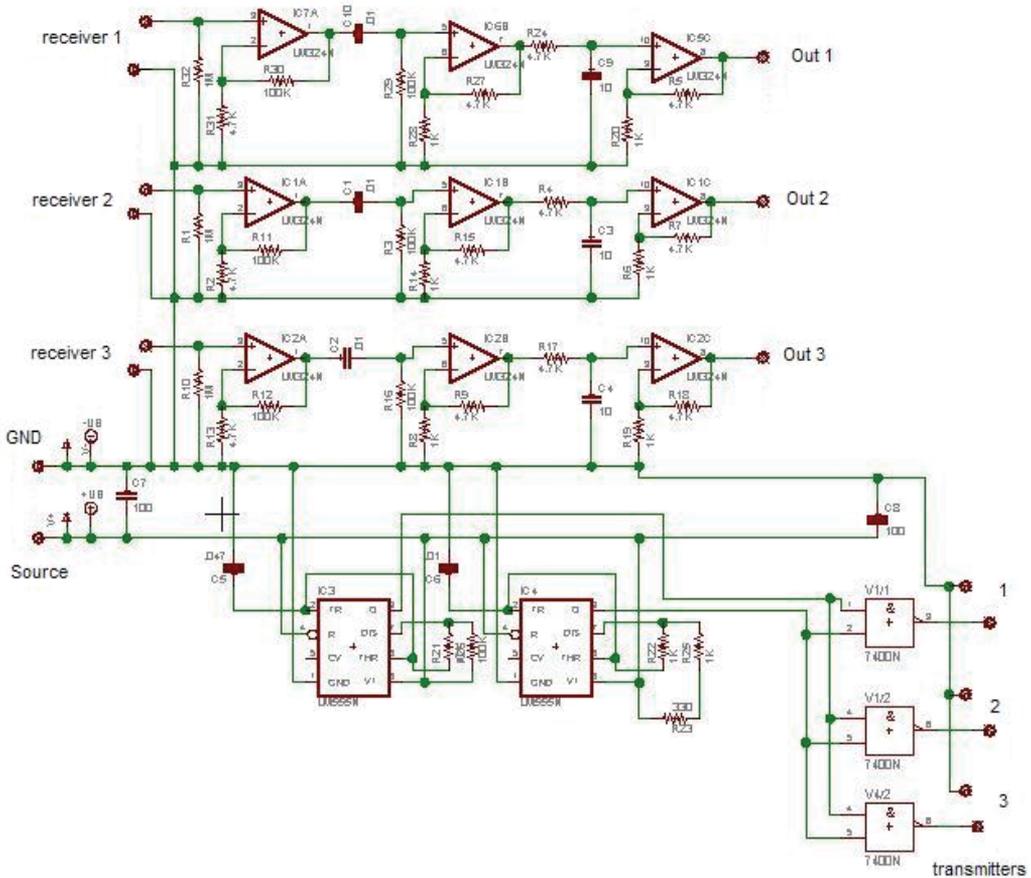


Figure 3. Schematics of the extension circuit of the ultrasonic sensors

The receptor circuit is made of three amplified phases: the first one amplifies the signal, within the frequency response range of the OpAm, at a maximum gain of 21 Volts/Volts; the second phase is a high-pass filter that cuts low frequencies down to 40 KHz and boosts the carrier signal by a factor of 4.7, which is enough to detect an increase in the proximity of detected objects; finally, the third phase is a low-pass filter that eliminates the carrier signal of 40 KHz to obtain a variation of the object-proximity signal within a range of 0 to 5 Volts that provides a good accuracy in the detection of obstacles within a maximum range of 20 centimeters. The ultrasonic sensor has a response-frequency that facilitates the detection of the 40 KHz carrier signal; therefore, noise from the environment is ignored. The construction of the extension board required that the design was implemented onto a single physical board (Fig. 3).

Due to the small size of the components of the circuit board; all the circuits were fitted into a small circular area similar to that occupied by the Gen I/O turret (Fig. 4(a)). Therefore, the three ultrasonic sensors were fitted onto a single board (Fig. 4(b)) on top of the Khepera. The intensity in the readings of the ultrasonic sensors can be adjusted by the gains in the

Then, this vector is passed to the neural network, which is optimized by the genetic algorithm. Finally, the output of the network is scaled to ± 20 values for the DC motors. The weights of the neural network are transcribed using a direct encoding to the vector \mathbf{c} of 54 elements. Next, random initial values are generated for this vector \mathbf{c}_i , $-1 < c_i < 1$. The initial population G_0 is composed of $n = 100$ neural controllers. The two best individuals of a generation are copied as a form of *elitism*. *Tournament selection*, for each of the $(n/2)-1$ local competitions, produces two parents for breeding a new individual using a single random *crossover* point with a probability of 0.5. The new offspring is affected with a *mutation* probability of 0.01. Individuals of the new offspring are evaluated for about 7 seconds in the robot simulator. The fitness plot for this behavior is shown in Figure 5.

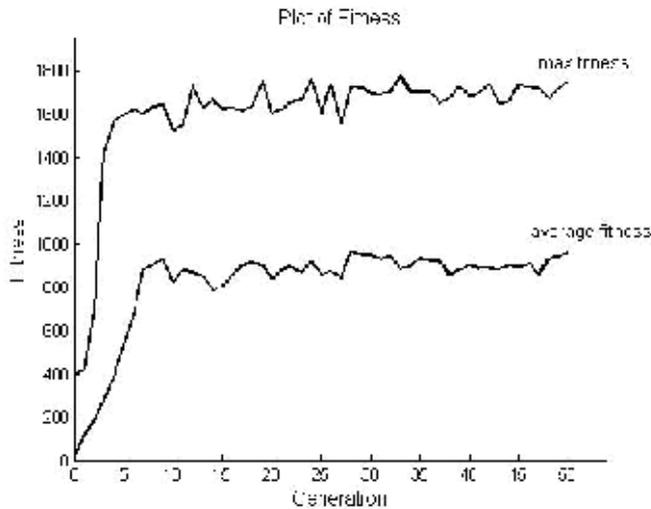


Figure 5. The fitness for the avoidance behavior is plotted over 50 generations

The fitness formula for the obstacle behavior was

$$f_q = \sum_{i=0}^{3500} abs(ls_i)(1 - \sqrt{ds_i})(1 - \max_sensor_i) \quad (2)$$

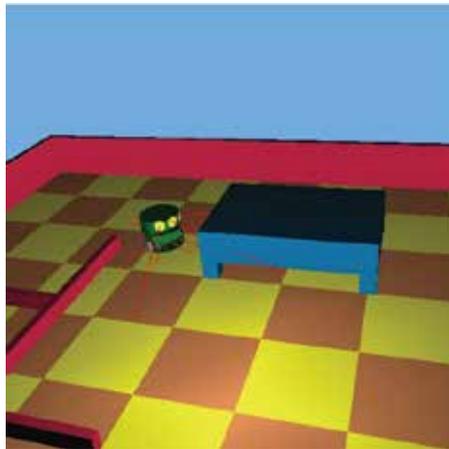
where for iteration i : ls is the linear speed in both wheels (the absolute value of the sum of the left and right speeds), ds is the differential speed on both wheels (a measurement of the angular speed), and \max_sensor is the maximum sensor value in the input vector. The use of a fitness formula like this rewards those fastest individuals who travel on a straight line while avoiding obstacles.

In our work we use a fitness formula similar to those of Floreano and Mondana (Floreano & Mondana, 1994) and Montes-Gonzalez (Montes, 2007). In a formula such as this the fitness is measured based on motor speeds and the obstruction of object-detection sensors. Therefore, an increase of the sensory capabilities of the robot may not be reflected in the fitness calculation. Since the addition of extra parameters, for including the information from infrared and sonar sensors may be redundant, it could affect the final selection of the fittest individuals. Therefore, we decided not to use any extra parameters.

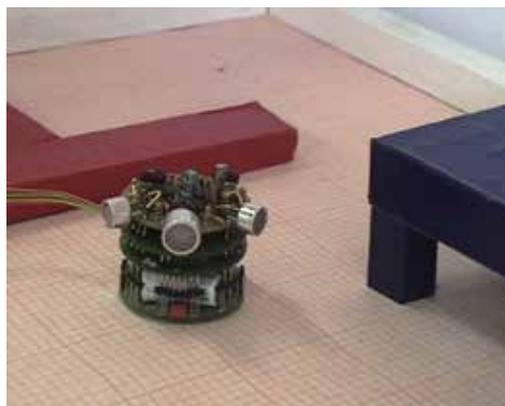
5. Experiments with the Khepera using Ultrasonic and Sonar Sensors

The transference of the neural controller to the Khepera required a hybrid approach that combines the use of a robot simulator (Fig. 6(a)) and the real robot (Fig. 6(b)). Usually the optimization of the neural network in the robot simulator Webots (Webots, 2006) lasted about a day. Then, the network is transferred to the robot, and the input scaled to appropriate values to compensate the artificial sensor readings of the simulator. It is important to notice that a hybrid approach is faster than a full-time optimization in the real robot.

The obstacle avoidance behavior is guided by the adjustment of the neural weights after optimization, and for evaluating the neural controllers we set the robot in a trial arena. Four walls define the borders of the arena, and short walls in the arena define a maze-like environment. A short-legged table is set inside the arena. Despite the space between the legs of the table and the floor not being detected by the infrared sensors; the Khepera has to avoid all kinds of obstacles. In Figure 6(b), we show the final setup for the real robot: an arena surrounded with white tall-walls, a blue table inside of the arena, and obstacles scattered around the arena as red short barriers.



a) Simulated Robot



b) Khepera Robot

Figure 6. The arena where the avoidance behavior occurs

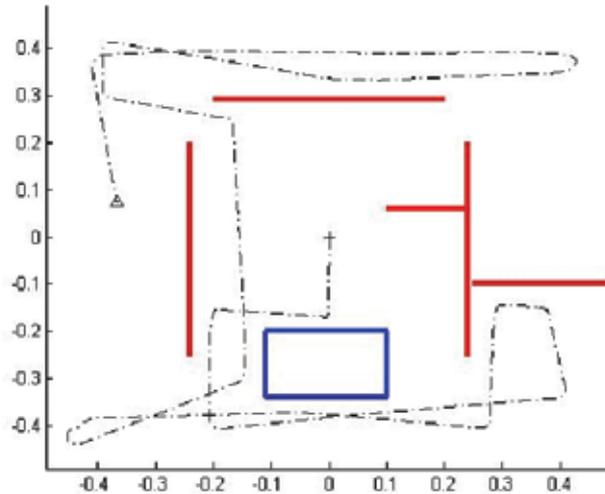


Figure 7. The plot of the trail of the robot in the arena with short obstacles and the table set inside the arena. Red Lines represent short walls that are mostly detected by the infrared sensors. On the other hand, the blue lines form a rectangular shape, which is the table with short legs that can only be detected with the use of the sonars. The dashed line shows the trail of the robot when running in the arena. The cross indicates the start of the avoidance behavior, and the triangle the stop of this behavior

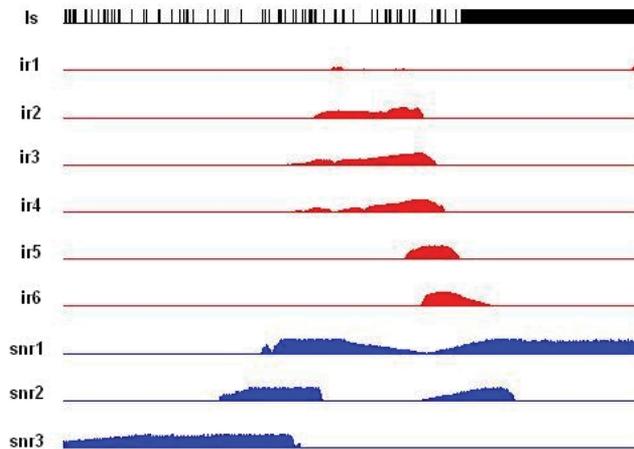


Figure 8. Plot of the infrared and the sonar sensor readings with the output of the linear motor speed. The black output represents the linear speed labeled as *ls*, this linear speed is calculated as in the fitness formula 2 and shows the robot running forward (a suppression of the signal represents a turning maneuver). The six frontal infrared sensors (*ir1* to *ir6*) are shown in red and indicate the presence of a close obstacle in front of the sensors. The sonars (*snr1* to *snr3*) are indicated in blue and represent activity when either a tall or legged object is near

The use of the simulator required that the sonars were modeled as if they drew a single ray proximity rather than a beam cone projection. However, as can be noticed in Figure 2(b) the frequency response of the sonar produces a peak where appropriate detection occurs. Therefore, the simplification of the sonar as a simple ray projection is consistent with the measured distance as the intensity of the bounced-back sound. The avoidance behavior is shown in Figure 7, where the plot of the trail of the robot is shown. Additionally, the plot of the Khepera motor output and its sensor-readings resume this avoidance behavior (Fig. 8)

The linear speed output [as in 2], the infrared sensors output, and the sonar-readings are plotted for about 500 iterations of the trial of the robot in the arena. It is important to notice that the forward traveling of the robot is canceled when a turning maneuver is done, then the detection of objects to be avoided is accomplished by the ring of infrared and sonar sensors. In Figure 8 we observe that the second half of the graph presents the activation of one of the side sonars, and even though this occurs, the robot keeps moving forward because it is running aside of one of the tall walls.

6. The Development of a Foraging Task with Action Selection

In the previous section, we presented how the capabilities of the Khepera can be extended to avoid a maze-like environment with obstacles having short-legs. The use of evolution facilitated the development of an avoidance behavior similar to that shown for a robot, set in an open arena, finding a safe position next to walls. However, we need to explain first how this avoidance behavioral-pattern can be set within an action selection system. The presence of an arbitration scheme within a functional system, where sub-systems exist that compete for gaining control of some associated shared resources, can be identified as an action selection mechanism that solves the action selection problem. Different models have been proposed to design systems, which are able to exhibit a variety of behavior and to arbitrate between them (e.g. behavior based architecture Brooks (1986), Arkin (1998)). Nevertheless, these models based on explicit design does not seem to be scalable enough for being applied to the development of systems capable of displaying a large variety of behavioral patterns that cope with task/environmental variations. One model that is modular and able to cope with the variations of a dynamic environment is CASSF (Central Action Model with Sensor Fusion). A complete description of the implementation of a foraging behavior using this model can be found in the work of Montes-Gonzalez & Marín-Hernández (2004). In this study we are focusing on the development of the avoidance behavior in non-homogeneous maze-like environment. Once the avoidance behavioral pattern has been developed, this can be added to the behavioral repertoire of CASSF (Figure 9).

Additionally, CASSF can be set within a foraging framework with evolved behavior (Montes-Gonzalez, et al., 2006a). On this implementation CASSF employs five basic behaviors that are: *cylinder-seek*, *cylinder-pickup*, *wall-seek*, *cylinder-deposit* and *look-around*. These behaviors can be described as follows: *cylinder-seek* explores the arena searching for food while avoiding obstacles; *cylinder-pickup* clears the space for collecting cylinders; *wall-seek* locates walls while avoiding obstacles; *cylinder-deposit* lowers and opens an occupied gripper; then *look-around* makes a full spin of the robot trying to locate the nearest perceptible cylinder. The development of a foraging framework similar to this one should be sufficient to complete the same task with the substitution of the avoidance

behavior in *wall-seek* by the avoidance behavioral pattern described in this work. As a consequence, collection of cylinders in an arena with short-walls can be completed using an improved avoidance-behavioral pattern.

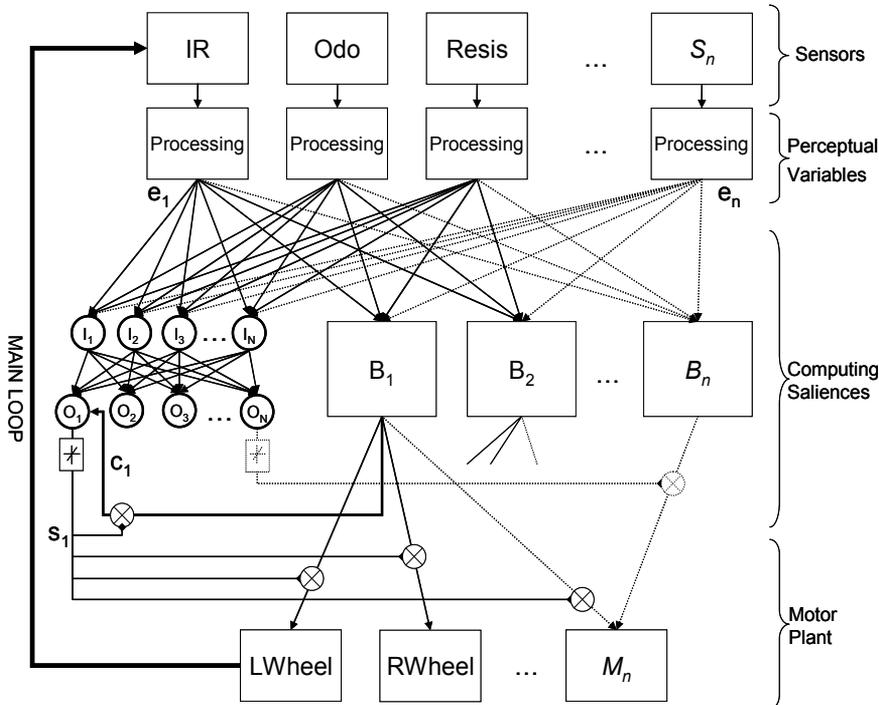


Figure 9. In the CASFF model, perceptual variables (e_i) form the input to the decision neural network. The output of the selected behavior with the highest salience (s_i) is gated to the motors of the Khepera. The busy-status signal (c_1) from behavior B1 to the output neuron O1 should be noticed. The repertoire of behaviors B_1 to B_n can be extended in CASSF by preserving similar connections for each of the additional behaviors that are added. Furthermore, behaviors can be improved off-line and then reinstalled back into the behavioral sub-system

A model such as CASSF is an effective Action Selection Mechanism (Montes-Gonzalez, et al., 2006b) that is centralized and presents sufficient persistence to complete a task. Furthermore, selection parameters of this model have been optimized by the use of evolution. The adjustment of selection parameters and behavior has been optimized by co-evolution in CASSF as described in (Montes-Gonzalez, 2007). The implementation of a foraging framework can be carried out in CASSF by determining some behavioral patterns that can be integrated in time to complete such a task. However, on this work we intend to focus on how the standard Khepera architecture can be improved by the use of evolution and the addition of extra sensors. Therefore, the foraging task has been explained in a concise manner. Next, we draw a general discussion on the further development of these experiments.

7. General Discussion

The use of sonars facilitates the detection of objects not in the direct vicinity of the robot. The sonars send out an ultrasound signal that is reflected back when an object is reached. An elegant solution to estimate the distance of an object employs the Khepera's microcontroller to measure the return time on an ultrasound signal. However, the estimation of distance following this approach is slow, expensive and still requires the use of the General I/O extension turret. In this study a better solution is provided by fitting three ultrasonic sensors on top of the Gen I/O turret, then distance is estimated from the intensity of the reflected sound. The information of the sonars is combined with that of the infrared sensors; then the input signals of both types of sensors are processed using a neural controller with optimized weights by means of a genetic algorithm. In order to speed up the optimization time a robot simulator is employed and the behavior is finally transferred with minor adjustments to the real robot. For the detection of objects like a short-legged table, the robot presents difficulties for the perception of this type of objects relying only on the infrared sensors. Therefore, the improved perception of the Khepera is able to locate non-homogenous collision objects that need to be avoided. The avoidance behavior with an improved perception can be included within a foraging behavioral framework to complete this task by maneuvering in an environment where non-homogenous-collision objects have been scattered around the arena. Additionally, the work presented here can be extended by the optimization of the neural network topology with the use of the genetic algorithm. Finally, it is our intention to expand these experiments by including a foraging prey that is able to collect food-items and run below obstacles like the table also a predator that has to follow the prey avoiding all kinds of obstacles.

8. Acknowledgments

This work has been sponsored by CONACyT-MEXICO grant SEP-2004-C01-45726.

9. References

- Airmar-Technology-Corporation (2006). *Specifications Data-sheet for the A30 Ultrasonic Sensor*, <http://airmar.com/>.
- Arkin, R. C. (1998). *Behavior-Based Robotics*. MIT Press, ISBN 9780262011655, USA.
- Bajaj, D. & Ang Jr., M. H. (2000). *An Incremental Approach in Evolving Robot Behavior, Presented at the Sixth International Conference on Control, Automation (ICARCV'2000), Robotics and Vision*, CD-ROM with ISBN 9810434456, Singapore, DEC 2000, Nanyang Technological University, Singapore.
- Böndel, D., Grünewald, M., Hanna, T. & Sandmeier, F. (1999). *Collective Cube Clustering using a specially designed hardware add-on*, In: *Proceedings of the 1st International Khepera Workshop*, Löffler Axel, Mondana Francesco and Rückert Ulrich (Eds.), pp. 253-254, Heinz Nixdorf Institut Paderborn Universität, ISBN 3931466639, Paderborn, Germany.
- Brooks, R. A. (1986). *A Robust Layered Control System For A Mobile Robot*, *IEEE Journal Of Robotics And Automation*, Vol. 2, Issues 1, MAR 1986, pp. 14-23, ISSN 08824967.

- Chapman, T., Hayes, A. & Tilden, M. (2000). Reactive Maze Solving with Biologically-Inspired Wind Sensor. In: *From Animals to Animats 6: Proceedings of the 6th International Conference on the Simulation of Adaptive Behavior*, Berthoz A., Floreano D., Meyer J., Roitblat H. and Wilson S. (Eds.), pp. 81-87, ISBN 0262632004, France, AGO 2000, MIT Press, Paris.
- Floreano, D. & Mondana, F. (1994). Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robot, In: *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, D. Cliff, P. Husbands, J.-A. Meyer, and S. Wilson (Eds.), pp. 421-430, MIT Press, ISBN 0262531224, Brighton, England.
- Goerke, N., Kintzler, F., Raabe, A. & Roggisch, D. (1999). Golf Playing Khepera, In: *Proceedings of the 1st International Khepera Workshop*, Löffler Axel, Mondana Francesco and Rückert Ulrich (Eds.), pp. 245-246, Heinz Nixdorf Institut Paderborn Universität, ISBN 3931466639, Paderborn, Germany.
- Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, ISBN 0262581116, Ann Arbor.
- Mondana, F., Franzi, E. & Paolo, I. (1993). Mobile robot miniaturisation: A tool for investigating in control algorithms, *Presented at the Proceedings of the 3rd International Symposium on Experimental Robotics III LNCIS Vol. 200*, T.Y. Yoshikawa & F. Miyazaki (Eds.), pp. 501-513, ISBN 3540199055, Japan, OCT 1993, Springer-Verlag, Kyoto.
- Montes-González, F. M., and A. Marín-Hernández (2004), Central Action Selection using Sensor Fusion. *Presented at the Proceedings of the Fifth Mexican International Conference in Computer Science (ENC'04)*, Baeza-Yates R., Marroquín J.L. and Chávez E (Eds.), pp. 289-296, ISBN 0-7695-2160-6, Colima, SEP 2004, IEEE Press, Mexico.
- Montes-González, F., J. Santos Reyes, and H. Ríos Figueroa (2006a), Integration of Evolution with a Robot Action Selection Model, *Presented at the Mexican International Conference on Artificial Intelligence (MICA I 2006)*, LNAI 4293, A. Gelbukh and C. A. Reyes-García (Eds.), pp. 1160-1170, ISBN 9783540490265, Apizaco, NOV 2006, Springer Berlin / Heidelberg, Mexico.
- Montes-González, F. M., A. Marín Hernández, and H. Ríos Figueroa (2006b), An Effective Robotic Model of Action Selection, In: *CAEPIA 2005*, LNAI 4177, R. Marín, et al. (Eds.), pp. 123-132, ISBN 9783540459149, Santiago de Compostela, NOV 2005, Springer Berlin / Heidelberg, Spain.
- Montes-Gonzalez, F. (2007). The Coevolution of Robot Behavior and Central Action Selection, In: *LNCIS 4528 Nature Inspired Problem-Solving Methods in Knowledge Engineering IWINAC 2007 Part II*, J. Mira and J.R. Álvarez (Eds.), pp. 439-448, Springer Berlin / Heidelberg, ISBN 9783540730545, La Manga del Mar Menor, Spain.
- Nolfi, S. & Floreano, D. (2000) . *Evolutionary Robotics*. The MIT Press, ISBN 0262140705, USA.
- Odenbach, C., Rüping, S., Löffler, A. & Rückert, U. (1999). An operating wireless CAN Communication System for Khepera Robots, In: *Proceedings of the 1st International Khepera Workshop*, Löffler Axel, Mondana Francesco and Rückert Ulrich (Eds.), pp. 181-187, Heinz Nixdorf Institut Paderborn Universität, ISBN 3931466639, Paderborn, Germany.

- Torben-Nielsen, B., Webb, B. & Reeve, R. (2005). New ears for a robot cricket. Lecture notes in computer science, Vol., Num 3696, SEP 2005, pp. 297-304, ISSN 03029743.
- Webots (2006). *Commercial Mobile Robot Simulation Software*. <http://www.cyberbotics.com>.
- Winge, F. (2004). *Sonar for a Khepera*. Dept. of Computer Science, Master Thesis, Lund University.

Multi-Legged Robot Control Using GA-Based Q-Learning Method With Neighboring Crossover

Tadahiko Murata^{1,2} and Masatoshi Yamaguchi³

¹ Policy Grid Computing Laboratory, Kansai University

² Department of Informatics, Kansai University

³ Space Systems & Public Information Systems Division, NEC Aerospace Systems, Ltd.
Japan

1. Introduction

Recently reinforcement learning has received much attention as a learning method (Sutton, 1988; Watkins & Dayan, 1992). It does not need a priori knowledge and has higher capability of reactive and adaptive behaviors. However, there are some significant problems in applying it to real problems. Some of them are deep cost of learning and large size of action-state space. The Q-learning (Watkins & Dayan, 1992), known as one of effective reinforcement learning, has difficulty in accomplishing learning tasks when the size of action-state space is large. Therefore the application of the usual Q-learning is restricted to simple tasks with the small action-state space. Due to the large action-state space, it is difficult to apply the Q-learning directly to real problems such as control problem for robots with many redundant degrees of freedom.

In order to cope with such difficulty of large action-state space, various structural and dividing algorithms of the action-state space were proposed (Holland, 1986; Svinin *et al.*, 2001; Yamada *et al.*, 2001). In the dividing algorithm, the state space is divided dynamically, however, the action space is fixed so that it is impossible to apply the algorithm to the task with large action space. In the classifier system, "don't care" attribute is introduced in order to create general rules. But, that causes the partially observable problem. Furthermore, an ensemble system of general and special rules should be prepared in advance.

Considering these points, Ito & Matsuno (2002) have proposed a GA-based Q-learning method called "Q-learning with Dynamic Structuring of Exploration Space Based on Genetic Algorithm (QDSEGA)." In their algorithm, a genetic algorithm is employed to reconstruct an action-state space which is learned by Q-learning. That is, the size of the action-state space is reduced by the genetic algorithm in order to apply Q-learning to the learning process of that space. They applied their algorithm to a control problem of multi-legged robot which has many redundant degrees of freedom and large action-state space. By applying their restriction method for action-state space, they successfully obtained the control rules for a multi-legged robot by their QDSEGA. However, the way to apply a genetic algorithm seems so straightforward in their study. Therefore we propose a crossover and a modified fitness definition for QDSEGA (Murata & Yamaguchi, 2005; Murata & Aoki,

2007). Through our computer simulations on a control problem of a multi-legged robot, we could make about 50% reduction of the number of generations to obtain a target state of the problem. In Murata & Aoki (2007), the same proposed crossover are used for controlling multiple agents who convey several loads to a goal. The proposed crossover is effective to decrease the number of actions to attain the objective. In this chapter, we concentrate on showing a QDSEGA with our crossover for controlling multi-legged robot.

In the previous study (Ito & Matsuno, 2002), the target of a multi-legged robot was fixed. That is the target does not move in their computer simulation. In this chapter, we try to move the target, and modify the learning algorithm of QDSEGA to follow the moving target. Simulation results show that our proposed method can control the multi-legged robot to follow the moving target more than the conventional method.

2. QDSEGA

In this section, we briefly explain the outline of QDSEGA (Ito & Matsuno, 2002). QDSEGA has two dynamics. One is a learning dynamics based on Q-learning and the other is a structural dynamics based on Genetic Algorithm. Fig. 1 shows the outline of QDSEGA. In QDSEGA, each action is represented by an individual of a genetic algorithm. According to actions defined by a set of individuals, action-state space called Q-table is created. Q-learning is applied to the created Q-table. Then the learned Q-table is evaluated through experiments. A fitness value for each action is assigned according to Q-table. After that, each individual (i.e., each action) is modified through genetic operations such as crossover and mutation. We show some details in these steps in the following subsections, and show our proposed method in the next section.

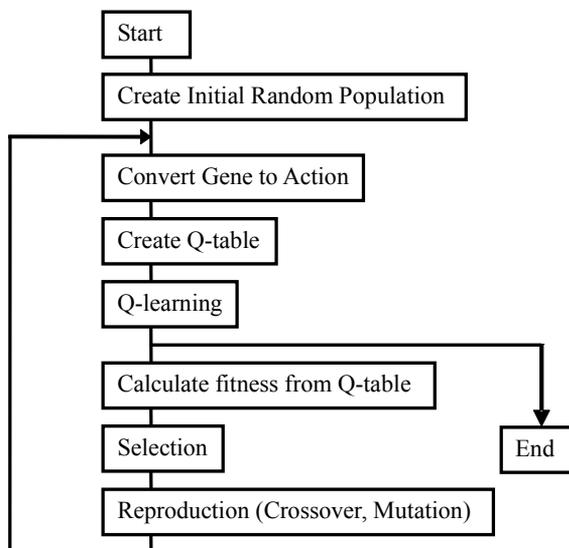


Figure 1. Outline of QDSEGA (Ito & Matsuno, 2002)

2.1 Action Encoding

Each individual expresses the selectable action on the learning dynamics. It means that a set of individuals is selected by genetic operations, and a learning dynamics is applied to the subset. After the evaluation of the subset of actions, a new subset is restructured by genetic operations.

2.2 Q-Table

An action-state space called Q-table is created from the set of individuals. When several individuals are the same code, that is, they are the same action, only one action is used in the action-state space in order to avoid the redundancy of actions. Fig. 2 shows this avoidance process.

2.3 Learning Dynamics

In QDSEGA, the conventional Q-learning (Watkins & Dayan, 1992) is employed as a learning dynamics. The dynamics of Q-learning are written as follows:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha\{r(s, a) + \gamma \max_{a'} Q(s', a')\}, \tag{1}$$

where $Q(s, a)$ is a Q-value of the state s and the action a , r is the reward, α is the learning rate, and γ is the discount rate.

2.4 Fitness

The fitness $fit(a_i)$ for each action is calculated by the following equation:

$$fit(a_i) = fit_Q(a_i) + k_f \cdot fit_u(a_i), \tag{2}$$

where $fit_Q(a_i)$ is a fitness value for action a_i calculated from Q-table, $fit_u(a_i)$ is a fitness value for action a_i calculated from the frequency of use, and k_f is a non-negative constant value to determine the ratio of $fit_Q(a_i)$ and $fit_u(a_i)$. We show the detail explanation of these factors in this subsection.

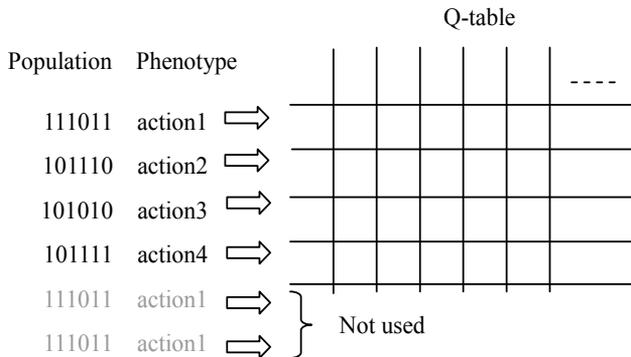


Figure 2. Q-table created from a set of individuals (Ito & Matsuno, 2002)

The fitness of Q-table $fit_Q(a_i)$ is calculated from Q-values in the current Q-table. In order to calculate $fit_Q(a_i)$ for each action a_i the following normalization is taken place in advance as for the Q-values in the current Q-table.

First, calculate the maximum and minimum value of each state as follows:

$$V_{\max}(s) = \max_{a'}(Q(s, a')), \quad (3)$$

$$V_{\min}(s) = \min_{a'}(Q(s, a')). \quad (4)$$

Then $Q'(s, a)$ of the normalized Q-table is given as follows:

If $Q(s, a) \geq 0$ then

$$Q'(s, a) = \frac{1-p}{V_{\max}(s)}Q(s, a) + p, \quad (5)$$

else $Q(s, a) < 0$ then

$$Q'(s, a) = -\frac{p}{V_{\min}(s)}Q(s, a) + p, \quad (6)$$

where p is a constant value which means the ratio of reward to penalty. After this normalization process, we fix the action a_i and sort $Q'(s, a_i)$ according to their value from high to low for all states. We define the sorted $Q'(s, a_i)$ as $Q'_s(s, a_i)$, and $Q'_s(1, a_i)$ means the maximum value of $Q'(s, a_i)$, and $Q'_s(N_s, a_i)$ means the minimum value of $Q'(s, a_i)$, where N_s is the size of states. Using the normalized and sorted Q-value $Q'_s(s, a_i)$, the fitness of action a_i is calculated as follows:

$$fit_Q(a_i) = \sum_{j=1}^{N_s} (w_j \frac{\sum_{k=1}^j Q'_s(k, a_i)}{j}), \quad (7)$$

where w_j is a weight which decides the ratio of special actions to general actions.

The fitness of frequency of use $fit_u(a_i)$ is introduced to save important actions. That fitness is defined as follows:

$$fit_u(a_i) = N_u(a_i) / \sum_{j=1}^{N_a} N_u(a_j), \quad (8)$$

where N_a is the number of all actions of one generation and $N_u(a_i)$ is the number of times which a_i was used for in the Q-learning of this generation. Important actions are used frequently. Therefore the actions with high fitness value of $fit_u(a_i)$ are preserved by this fitness value.

2.5 Genetic Algorithms

The paper that proposed QDSEGA (Ito & Matsuno, 2002) says "the method of the selection and reproduction is not main subject so the conventional method is used (in this paper)." They employed a crossover that exchanges randomly selected bits between the parent

individuals according to the crossover probability P_c . They mutated each bit according to the mutation probability P_m . They did not replace parents individuals with offspring. Therefore the number of individuals is increased by the genetic operations. As for the elite preserving strategy, they preserve 30% individuals with the highest fitness value.

3. Proposed Method

We propose a crossover operation for the multi-legged robot control problem (MRC problem). As shown in Subsection 2.5, QDSEGA did not try to propose any specific genetic operations for MRC problems. We propose a neighboring crossover for QDSEGA for MRC problems. In order to show our crossover which is tailored to MRC problems, we explain a coding method of a multi-legged robot and Q-table used in this chapter first.

3.1 Coding of Individuals

Fig. 3 shows the multi-legged robot and its encoding. We control a robot which has 12 legs. Each leg has four states as in Table 1. As shown in Fig. 3, each individual shows a set of legs' positions of 12 legs. Therefore each individual indicates one state of this robot of the 4^{12} positions. Since we generate only the prespecified number of individuals in genetic algorithms, the Q-table includes a subset of 4^{12} states with respect to legs' positions.

3.2 Q-Table for 12-Legged Robot

Q-table represents a subset of the action-state space to control a 12-legged robot. In QDSEGA, individuals are used for the both of actions and states. Fig. 4 shows an example of a Q-table created by n individuals. In this problem, each individual is used twice in the Q-table. "Current state" indicates the current position of 12 legs. "Action" indicates a next position of 12 legs after a certain action. For example, if S1 is the current position of 12 legs and A3 is selected as an action, the 12-legged robot moves its 12 legs to become the next position as A3. If the action A3 causes the robot to lose balance, the trial terminates. In our simulation, we assume that the 12-legged robot can change its legs' position smoothly.

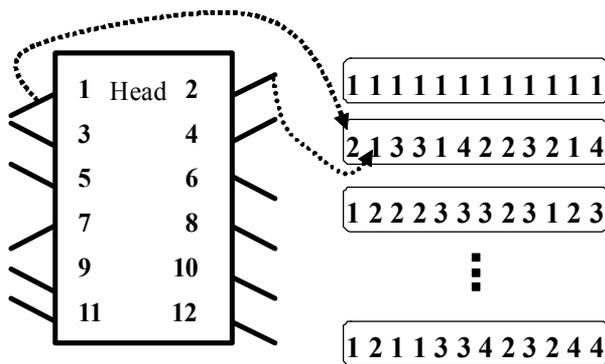
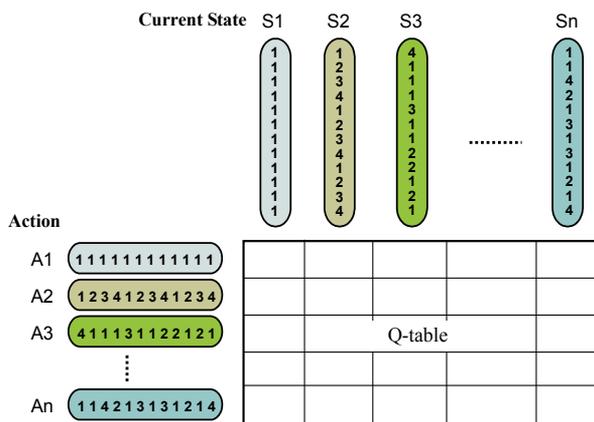


Figure 3. 12 Legs and representation in an individual

State Number	State
1	Forward and grounded
2	Backward and grounded
3	Forward and ungrounded
4	Backward and ungrounded

Table 1. Leg's position

Figure 4. An example of Q-table created by n individuals

3.3 Neighboring Crossover

The crossover employed in QDSEGA (Ito & Matsuno, 2002) causes drastic change of legs' position because randomly selected bits are changed between two individuals. If the change of legs' position is so drastic, it may easily happen that the 12-legged robot loses its balance. In order to avoid losing the balance, we propose a crossover between similar parent individuals. We define the similarity by the number of the same genes in the same locus of a chromosome. The threshold for the similarity between two parents is denoted by k_{sim} in this chapter. Thus, the crossover is applied among similar individuals more than k_{sim} .

This kind of the restriction for the crossover has been proposed in the research area of distributed genetic algorithms. Researches on DGAs (Distributed Genetic Algorithm) can be categorized into two areas: coarse-grained genetic algorithms (Tanese, 1989; Belding, 1995) and fine-grained genetic algorithms (Mandelick & Spiessens, 1989; Muhlenbein *et al.*, 1991; Murata *et al.*, 2000). In the coarse-grained GAs, a population, that is ordinarily a single, is divided into several subpopulations. Each of these subpopulations is individually governed by genetic operations such as crossover and mutation, and subpopulations communicate each other periodically. Algorithms in this type are called the island model because each subpopulation can be regarded as an island. On the other hand, several individuals are locally governed by genetic operations in fine-grained GAs. In a fine-grained GA, each individual exists in a cell, and genetic operations are applied to an individual with individuals in neighboring cells. The DGAs are known to have an advantage to keep the variety of individuals during the execution of an algorithm, and avoid converging prematurely.

While we don't define any solution space such as cells or islands in our proposed crossover, our restriction may have the same effect of keeping variety in a population and attain the effective search for the sequence of states of 12 legs.

3.4 Modified Reward for Moving Target

In the previous study, the target of the MRC problem was fixed. That is, the target stays at the same place. We consider a moving target as a variant problem of MRC problems. In order to let the multi-legged robot follow a moving target, we modify the reward $r(s, a)$ at time t in Equation (1) as follows:

If $-45[\text{deg}] \leq L_z < -15[\text{deg}]$ and $d(t-1) - d(t) \geq 0$ then

$$r(s, a) = 50 \cdot (d(t-1) - d(t)), \quad (9)$$

If $-45[\text{deg}] \leq L_z < -15[\text{deg}]$ and $d(t-1) - d(t) < 0$ then

$$r(s, a) = 200 \cdot (d(t-1) - d(t)), \quad (10)$$

If $15[\text{deg}] < L_z \leq 45[\text{deg}]$ and $d(t-1) - d(t) \geq 0$ then

$$r(s, a) = 50 \cdot (d(t-1) - d(t)), \quad (11)$$

If $15[\text{deg}] < L_z \leq 45[\text{deg}]$ and $d(t-1) - d(t) < 0$ then

$$r(s, a) = 200 \cdot (d(t-1) - d(t)), \quad (12)$$

If $-15[\text{deg}] \leq L_z \leq 15[\text{deg}]$ then

$$r(s, a) = 100 \cdot (d(t-1) - d(t)), \quad (13)$$

Else If ($L_z < -45[\text{deg}]$ or $45[\text{deg}] < L_z$ or stuck) then

$$r(s, a) = -100. \quad (14)$$

In the above equations, L_z is the direction of the target in degrees (not in radian), and $d(t)$ is the distance from the head of the robot to the target at the time t (see Fig. 5 for the detail). L_z is zero [deg] if the target locates just ahead of the robot.

4. Multi-Legged Robot Control

4.1 Task

Fig. 5 shows the initial position of the 12-legged robot and the location of the target. At time zero, the head of the robot locates in (0,0) in the x - y plane. And the initial state of each leg of the robot is 1 in Table 1. We allow the robot to move 100 steps toward the target. If the robot can acquire the most efficient movement of 12 legs, it can move 0.4 for each step. Therefore the maximum gain toward the target is 40.

When we try to acquire the legs' movement for the moving target, the robot first learns the legs' movement for the fixed target over 50 generation. After that we put the target at $(100\sqrt{2}, -100\sqrt{2})$ in the x - y plane, and move it to $(100\sqrt{2}, 100\sqrt{2})$ by 1 [deg] in each step

on the circle whose center is (0,0) and the radius is 200. After 90 steps, the target reaches the point $(100\sqrt{2}, 100\sqrt{2})$ and stay there until the end of 100 steps in each trial.

4.2 Simulation Model for Multi-Legged Robot

As Ito & Matsuno (2002) did, we also employed Minimal Simulation Model that was proposed by Svinin *et al.* (2001). This model is very simple so that the calculation cost becomes very low.

Fig. 6 shows a multi-legged robot. Each leg has two joints and has four motions. (Move forward and lift down, Move back and lift down, Move forward and lift up, Move back and lift up). The position of the robot can be calculated as follows:

$$f_{drv}^r = u(n_{12}^r - n_{21}^r), \tag{15}$$

$$f_{res}^r = v(n_{11}^r + n_{22}^r), \tag{16}$$

$$\text{if } |f_{drv}^r| \leq f_{res}^r \text{ then } F^r = 0, \tag{17}$$

$$\text{else if } |f_{drv}^r| > f_{res}^r \text{ then } F^r = f_{drv}^r - f_{res}^r, \tag{18}$$

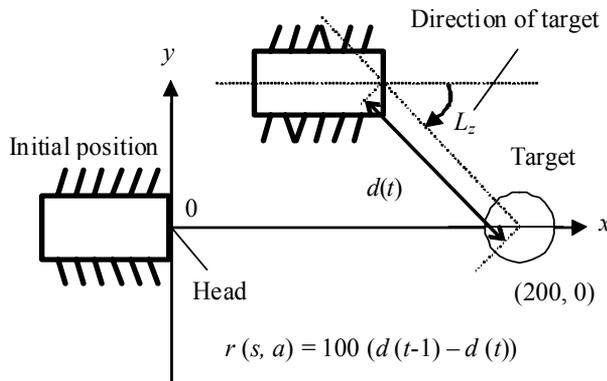


Figure 5. Initial position and the target (Ito & Matsuno, 2002)

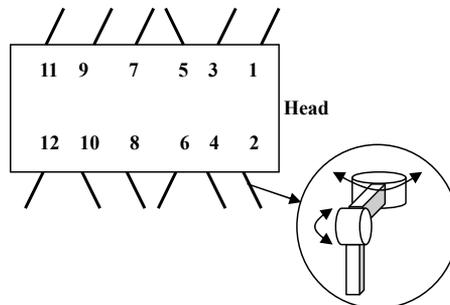


Figure 6. Multi-legged robot (Ito & Matsuno, 2002)

$$\text{else } f_{drv}^r < -f_{res}^r \text{ then } F^r = f_{drv}^r + f_{res}^r, \quad (19)$$

$$F = F^l + F^r, \quad (20)$$

$$M = F^r - F^l, \quad (21)$$

$$\Delta u = c_u F, \quad (22)$$

$$\Delta \theta = c_\theta M, \quad (23)$$

$$x(t+1) = x(t) + \Delta u \cos \theta(t), \quad (24)$$

$$y(t+1) = y(t) + \Delta u \sin \theta(t), \quad (25)$$

$$\theta(t+1) = \theta(t) + \Delta \theta, \quad (26)$$

where f_{drv}^r and f_{res}^r indicate driving force and resistance force, and n_{ij}^r is the number of legs on the right side changing their configuration from the state i at the moment t to the state j at the moment $t + 1$. Similarly, we define n_{ij}^l for the left side. Let F and M be the force and the moment of the robot that act to the environment, respectively. And x, y, θ mean the position and orientation of the robot. Details are written in Svirin *et al.* (2001).

5. Simulation

5.1 Parameter Settings

We specified the parameter settings in the Q-learning, the genetic algorithm, and the multi-legged robot simulation model as follows:

[Q-learning]

The number of trials for each Q-table: 3 000 trials,

The number of steps in each trial: 100 steps,

Learning rate in (1): $\alpha = 0.7$,

Discounting rate in (1): $\gamma = 0.7$,

Temperature of Boltzmann distribution: $T = 150 \times e^{(-0.002 \times \text{trial}) + 1}$.

[Genetic Algorithm]

The number of individuals: 50,

The number of generations: 200,

Crossover probability: $P_c = 0.5$,

Similarity for the crossover: $k_{\text{sim}} = 4$,

Mutation probability: $P_m = 0.2$,

Weight value in (2): $k_f = 200$,

Ratio of reward to penalty in (5), (6): $p = 0.01$,

Weight values in (7): $w_1 = 0.6$, $w_i = 0$ ($i = 2, \dots, N_s - 1$), $w_{N_s} = 0.4$.

[Multi-Legged Robot Simulator]

$u = 2.0$, $v = 1.0$, $c_\theta = 0.05$, $c_u = 0.05$.

5.2 Effect of Neighboring Crossover

We applied the conventional method (Ito & Matsuno, 2002) and the proposed method with the neighboring crossover in Subsection 3.3 100 times. Fig. 7 shows the average “Gain” over the generation. We defined “Gain” as the gaining distance of the robot toward the target. Thus, the maximum gain was 40 in this problem. From Fig. 7, we can see that the proposed crossover clearly enhanced the performance of QDSEGA.

Table 2 shows that the average number of generations in which the target gain was attained over 100 experiments. We specified the target gain as 38 in these experiments. We count a computational experiment as successful one when the robot gains over 38. We also show the standard deviation of the generations and the number of successful experiments. “Successful experiments” mean that the number of experiments that can attain the target gain. “Ito 2002” shows the conventional method in (Ito & Matsuno, 2002), and “Neighboring Crossover” is the method with the proposed neighboring crossover in Subsection 3.3. From Table 2, we can see that the number of generations was drastically reduced by introducing the neighboring crossover into QDSEGA. As shown in “# of successful experiments,” the conventional method could not attain the maximum gain in all the 100 experiments while “Neighboring Crossover” attained the maximum gain in all the experiments. Through this result we can see the strong effect of the proposed neighboring crossover in QDSEGA.

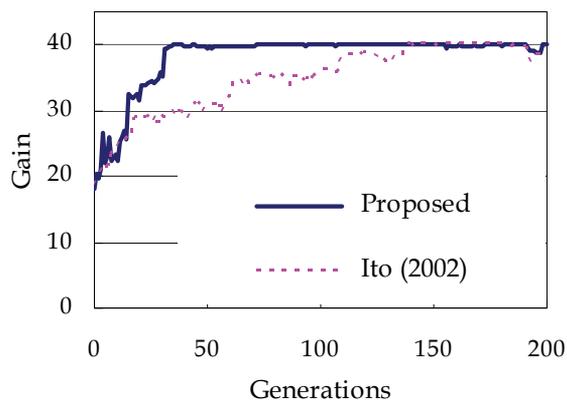


Figure 7. Average gain over the generation

	Ito (2002)	Neighboring Crossover
Average Generation	108.5	56.2
Standard Deviation	37.7	19.3
# of successful experiments	85/100	100/100

Table 2. Average generation of attaining the maximum gain over 100 experiments

Conventional	Without adjustment		
Target position	Left	Ahead	Right
Ratio (%)	21.7	50.7	27.5

Conventional	With adjustment		
Target position	Left	Ahead	Right
Ratio (%)	17.8	63.4	17.8

Table 3. Effect on the conventional method for moving target

Proposed	Without adjustment		
Target position	Left	Ahead	Right
Ratio (%)	21.0	50.6	28.2

Proposed	With adjustment		
Target position	Left	Ahead	Right
Ratio (%)	21.9	63.3	14.8

Table 4. Effect on the conventional method for moving target

5.3 Effect of Modified Reward for Moving Target

Tables 3 and 4 show the results on the MRC problem with the moving target. We applied our modified reward to QDSEGA (Ito & Matsuno, 2002) and QDSEGA with the proposed crossover. In these tables, "Target position" indicates that the ratio of target positions from the robot. That is, when the target locates ahead of the robot within -15 [deg] to 15 [deg], the number of "Ahead" is incremented. From this table, we can see that the ratio of "Ahead" was increased by the introduction of the adjustment of rewards in Q-learning in the both tables. Through the proposed modification of the reward we could improve the performance of the multi-legged robot to follow the moving target.

6. Conclusion

In this chapter, we proposed the neighboring crossover that improves the performance of QDSEGA for the multi-legged robot control problems. By introducing the neighboring crossover, we could reduce the number of generations to attain the target gain in the problem. We also show the fine effect of the modified reward for the problems with the moving target. When the target is moving in the problem, the robot needs to learn how to adjust its position toward the target. That is, it should learn how to adjust its direction toward the target, and how to convey its body toward the target. By adjusting the reward, we could improve the control of the multi-legged robot.

7. Acknowledgments

This work was partially supported by the MEXT, Japan under Collaboration with Local Communities Project for Private Universities starting 2005.

8. References

- Belding, T.C. (1995). The distributed genetic algorithm revisited, *Proc. of 6th International Conference on Genetic Algorithms*, pp. 114-121.
- Holland, J.H. (1986). Escaping brittleness: the possibilities of general purpose learning algorithms applied to parallel rulebased system. *Machine Learning II*, pp. 593-623.
- Ito, K. & Matsuno, F. (2002). A study of reinforcement learning for the robot with many degrees of freedom -Acquisition of locomotion patterns for multi legged robot-, *Proc. of IEEE International Conference on Robotics and Automation*, pp. 392-397.
- Mandelick, B. & Spiessens, P. (1989). Fine-grained parallel genetic algorithms, *Proc. of 3rd International Conference on Genetic Algorithms*, pp. 428-433.
- Muhlenbein, H. Schomisch, M. & Born, J. (1991). The parallel genetic algorithm as function optimizer, *Proc. of 4th Int'l Conf. on Genetic Algorithms*, pp. 271-278.
- Murata, T. & Aoki, Y. (2007). Developing Control Table for Multiple Agents Using GA-Based Q-Learning With Neighboring Crossover, *Proc. of IEEE Congress on Evolutionary Computation 2007*, pp. 1462-1467.
- Murata, T., Ishibuchi, H. & Gen, M. (2000). Cellular genetic local search for multi-objective optimization, *Proc. of Genetic and Evolutionary Computation Conference 2000*, pp. 307-314.
- Murata, T. & Yamaguchi M. (2005). Neighboring Crossover to Improve GA-Based Q-Learning Method for Multi-Legged Robot Control, *Proc. of Genetic and Evolutionary Computation 2005*, pp. 145-146.
- Sutton, R.S. (1988). *Reinforcement Learning: An Introduction*. The MIT Press.
- Svinin, M., Ushio, S., Yamada, K. & Ueda, K. (2001). An evolutionary approach to decentralized reinforcement learning for walking robots, *Proc. of 6th Int. Symp. on Artificial life and Robotics*, pp. 176-179.
- Tanese, R. (1989). Distributed genetic algorithms, *Proc. of 3rd Int'l Conf. on Genetic Algorithms*, pp. 434-439.
- Watkins, C.J.C.H. & Dayan, P. (1992). Technical note Q-learning, *Machine Learning*, Vol. 8, pp. 279-292.
- Yamada, K., Ohkura, K., Svinin, M. & Ueda, K. (2001). Adaptive segmentation of the state space based on bayesian discrimination in reinforcement learning, *Proc. of 6th International Symposium on Artificial life and Robotics*, pp. 168-171.

Evolved Navigation Control for Unmanned Aerial Vehicles

Gregory J. Barlow¹ and Choong K. Oh²

¹*Robotics Institute, Carnegie Mellon University*

²*United States Naval Research Laboratory
United States*

1. Introduction

Whether evolutionary robotics (ER) controllers evolve in simulation or on real robots, real-world performance is the true test of an evolved controller. Controllers must overcome the noise inherent in real environments to operate robots efficiently and safely. To prevent a poorly performing controller from damaging a vehicle—susceptible vehicles include statically unstable walking robots, flying vehicles, and underwater vehicles—it is necessary to test evolved controllers extensively in simulation before transferring them to real robots. In this paper, we present our approach to evolving behavioral navigation controllers for fixed wing unmanned aerial vehicles (UAVs) using multi-objective genetic programming (GP), choosing the most robust evolved controller, and assuring controller performance prior to real flight tests.

2. Background

ER (Nolfi & Floreano, 2000) combines robot controller design with evolutionary computation. A major focus of ER is the automatic design of behavioral controllers with no internal environmental model, in which effector outputs are a direct function of sensor inputs (Keymeulen et al., 1998). ER uses a population-based evolutionary algorithm to evolve autonomous robot controllers for a target task. Most of the controllers evolved in ER research have been developed for simple behaviors, such as obstacle avoidance (Nolfi et al., 1994), light seeking (Lund & Hallam, 1997), object movement (Lee & Hallam, 1999), simple navigation (Ebner, 1998), and game playing (Nelson, 2003; Nelson et al., 2003). In many of these cases, the problems to be solved were designed specifically for research purposes. While simple problems generally require a small number of behaviors, more complex real-world problems might require the coordination of multiple behaviors in order to achieve the goals of the problem. Very little ER work to date has been intended for use in real-life applications.

A majority of the research in ER has focused on wheeled mobile robot platforms, especially the Khepera robot. Research on walking robots (Filliat et al., 1999) and other specialized robots (Harvey et al., 1994) has also been pursued. An application of ER that has received very little attention is UAVs. The UAV has become popular for many applications, particularly where high risk or accessibility is concerns. Although some ER research has

been done on UAVs, this work has largely ignored the fixed wing UAV—by far the most common type—until recently. An autopilot for a rotary wing helicopter was evolved using evolutionary strategies (Hoffman et al., 1998) and compared to linear robust multi-variable control and nonlinear tracking control in simulation (Shim et al., 1998). In other work, higher level controllers were evolved with UAVs as the target platform (Marin et al., 1999), but experiments were done only in simulation, movement was grid-based, and the UAV could move in any direction at every time step. Because of the unrealistic nature of the simulation, it would have been difficult to control real UAVs with the evolved controllers. Related work was done to evolve a distributed control scheme for multiple micro air vehicles (Wu et al., 1999). Only simulation was used, the simulation environment was unrealistic, and no testing on real UAVs was attempted. A neural network control system for a simulated blimp has also been evolved (Meyer et al., 2003) with the goal of developing controllers capable of countering wind to maintain a constant flying speed. The evolved control system was only tested in simulation. Only recently has there been work on evolving GP controllers for fixed wing UAVs (Oh et al., 2004; Barlow, 2004; Oh & Barlow, 2004; Barlow et al., 2004; Barlow et al., 2005; Richards et al., 2005; Barlow & Oh, 2006).

In evolutionary computation, incremental evolution (Harvey et al., 1994) is the process of evolving a population on a simple problem and then using the resulting evolved population as a seed to evolve a solution to a related problem of greater complexity. Solutions to a variety of complicated problems in ER have been evolved using incremental evolution. There are two types of incremental evolution. Functional incremental evolution (Lee & Hallam, 1999; Gomez & Miikkulainen, 1997; Winkeler & Manjunath, 1998) changes the difficulty of the fitness function in order to increase the difficulty of the problem. Environmental incremental evolution (Harvey et al., 1994; Nelson, 2003; Nelson et al., 2003) changes the environment to increase difficulty without changing the fitness function.

Transference of controllers evolved in simulation to real vehicles is an important issue in ER. Some controllers have been evolved in situ on physical robots (Walker et al., 2003), but long evaluation time, the need for many evaluations to achieve good results, and the need for human monitoring during the evolutionary process all limit this approach. Alternatively, controllers evolved in simulation do not always transfer well to real vehicles, since the simulation is never a perfect model of the real environment. Adding noise to the simulation (in the form of both sensor error and state error) may help controllers transfer well from simulation to real robots (Jakobi et al., 1995; Gomez and Miikkulainen, 2004; Barlow et al., 2005). This approach is usually evaluated by evolving a controller in a noisy simulation environment and then testing the controller on a real vehicle. This works well for systems where tests can be performed easily, cheaply, and with little danger of damaging the vehicle, but what of systems where tests are expensive or dangerous? Controllers may be evolved with high levels of noise, but this does not guarantee good performance when that noise is not consistent with the real system. Experiments by Jakobi et al. (Jakobi et al., 1995) show that if the noise levels used in simulation are significantly different from those in the real world, there are no assurances that the evolved controller will perform as desired. If, however, a controller performs well when subjected to a wide range of sensor and state noise conditions in simulation, and the real environmental noise falls within the testing range, prior works suggest that the controller should also perform well on a real vehicle.

UAVs are one type of robot that requires assurance of the off-design performance (the performance under additional sensor and state noise) of an evolved controller before testing

a controller on the robot. Even when subject to additional sources of noise, controllers should still be able to efficiently accomplish the desired task. Assurance of off-design performance is also necessary because poorly performing controllers could cause crashes, possibly destroying the UAV.

3. UAV Navigation Control

The focus of this research was the development of a navigation controller for a fixed wing UAV able to autonomously locate, track, and then circle around a radar site. There are three main goals for an evolved controller. First, the UAV should move to the target as quickly as possible. The sooner the UAV arrives in the vicinity of the target, the sooner it can begin its primary mission: surveillance, radar jamming, or one of the many other applications of this type of controller. Second, once in the vicinity of the source, the UAV should circle as closely as possible around the radar. This goal is especially important for radar jamming, where the necessary jamming power is directly proportional to the square of the distance to the radar. Third, the flight path should be efficient and stable. The roll angle should change as infrequently as possible, and any change in roll angle should be small. Making frequent changes to the roll angle of the UAV could create dangerous flight dynamics or reduce the flying time and range of the UAV.

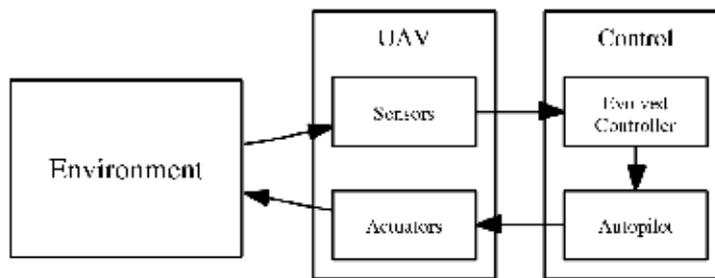


Figure 1. General UAV control diagram

Only the navigation portion of the flight controller is evolved; the low level flight control is done by an autopilot. The navigation controller receives radar electromagnetic emissions as input, and based on this sensory data and past information, the navigation controller updates the desired roll angle of the UAV control surface. The autopilot then uses this desired roll angle to change the heading of the UAV. A diagram of the control process is shown in Figure 1. This autonomous navigation technique results in a general controller model that can be applied to a wide variety of UAV platforms; the evolved controllers are not designed for any specific UAV airframe or autopilot.

3.1 Simulation

While there has been success in evolving controllers directly on real robots, simulation is the only feasible way to evolve controllers for UAVs. A UAV cannot be operated continuously for long enough to evolve a sufficiently competent controller, the use of an unfit controller could result in damage to the aircraft, and flight tests are very expensive. For these reasons, the simulation must be capable of evolving controllers which transfer well to real UAVs. A

method that has proved successful in this process is the addition of noise to the simulation (Jakobi et al., 1995).

The simulation environment is a square, 100 nautical miles (nmi) on each side. Every time a simulation is run, the simulator gives the UAV a random initial position in the middle half of the southern edge of the environment with an initial heading of due north. The radar site is also given a random position within the environment. In our current research, the UAV has a constant altitude of 3000 feet and speed of 80 knots. We can realistically assume constant speed and altitude because these variables are controlled by the autopilot, not the evolved navigation controller.

Our simulation can model a wide variety of radar types. The site type, emitter function, frequency, gain, noise, power, pulse compression gain, bandwidth, minimum emitting period, mean emitting period, minimum emitting duration, and mean emitting duration of the radar are all configurable in the simulation. For the purposes of this research, most of these parameters were held constant. Radars used in experiments are described based on two characteristics: emitting pattern and mobility. We modeled five types of radars: continuously emitting, stationary radars; continuously emitting, mobile radars; intermittently emitting, stationary radars with regular emitting periods; intermittently emitting, stationary radars with irregular emitting periods; and intermittently emitting, mobile radars with regular emitting periods.

Radars can emit continuously, intermittently with a regular period, or intermittently with an irregular period. The emitting characteristics of the radar are configured by setting the minimum emitting period, mean emitting period, minimum emitting duration, and mean emitting duration. If all four parameters are set to infinity, the radar is continuous. If the minimum and mean are the same for both period and duration, then the radar is considered to be emitting with a regular period. If the minimum and mean are different, the radar emits with an irregular period: at the start of each period, the lengths of the period and duration of emission are set randomly.

Radars can be either stationary or mobile. A stationary site has a fixed position for the entire simulation period. A mobile site is modeled by a finite state machine with the following states: *move*, *setup*, *deployed*, and *tear down*. When the radar moves, the new location is random, and can be anywhere in the simulation area. The finite state machine is executed for the duration of simulation. The radar site only emits when it is in the *deployed* state; while the radar is in the *move* state it does not emit, so the UAV receives no sensory information. The time in each state is probabilistic, and once the radar enters the *deployed* state, it must remain in that state for at least an hour.

Only the sidelobes of the radar emissions are modeled. The sidelobes of a radar signal have a much lower power than the main beam, making them harder to detect. However, the sidelobes exist in all directions, not just where the radar is pointed. This model is intended to increase the robustness of the system, so that the controller doesn't need to rely on a signal from the main beam. Additionally, Gaussian noise is added to the amplitude of the radar signal. The receiving sensor can perceive only two pieces of information: the amplitude and the angle of arrival (AoA) of incoming radar signals. The AoA measures the angle between the heading of the UAV and the source of incoming electromagnetic energy. Real AoA sensors do not have perfect accuracy in detecting radar signals, so the simulation models an inaccurate sensor. The accuracy of the AoA sensor can be set in the simulation. In the experiments described in this research, the AoA is accurate to within $\pm 10^\circ$ at each time

step, a realistic value for this type of sensor. Each experimental run simulates four hours of flight time, where the UAV is allowed to update its desired roll angle once a second, a realistic value for a real UAV autopilot. The interval between these requests to the autopilot can be adjusted in the simulation.

3.2 Transference

Transference of evolved controllers to a real UAV is an important issue, so we designed several aspects of the simulation to aid in this process. First, we abstracted the navigation control from the flight of the UAV. Rather than attempting to evolve direct control, only the navigation was evolved. This allows the same controller to be used for different airframes. Second, the simulation was designed so parameters could be tuned for equivalence to real aircraft and radars. For example, the simulated UAV is allowed to update the desired roll angle once per second, reflecting the update rate of the real autopilot of a UAV being considered for flight demonstrations of the evolved controller. For autopilots with slower response times, this parameter could be increased. Third, noise was added to the simulation, both to radar emissions and to sensor accuracy. A noisy simulation environment encourages the evolution of robust controllers that are more applicable to real UAVs.

3.3 Problem Difficulty

The major difficulty of this problem is noise. Under ideal conditions, where the exact angle and amplitude of the incoming signals are known, a human could easily design a fit controller. Real-world conditions, however, are far from ideal. Even the best radar sensors have some error in determining the angle and amplitude of a radar. Environmental conditions, multipath, system noise, clutter, and many other factors increase the sensor noise. As this noise increases, the difficulty of maintaining a stable and efficient flight path increases.

While sensors to detect the amplitude and angle of arriving electromagnetic signals can be very accurate, the more accurate the sensor, the larger and more expensive it tends to be. One of the great advantages of UAVs is their low cost, and the feasibility of using UAVs for many applications may also depend on keeping the cost of sensors low. By using evolution to design controllers, cheaper sensors with much lower accuracy can be used without a significant drop in performance.

Another difficulty of designing controllers by hand is accounting for the more complex radar types. As the accuracy of the sensors decreases and the complexity of the radar signals increases—as the radars emit periodically or move—the problem becomes far more difficult for human designers as the best control strategies become less apparent. In this research, we are interested in evolving controllers for these difficult, real-world problems using many radar types where sensors are very noisy.

3.4 Fitness Functions

We designed four fitness functions to measure the success of individual UAV navigation controllers. The fitness of a controller was measured over 30 simulation runs, where the initial positions of the UAV and the radar were different for every run. We designed the four fitness measures to satisfy the three goals of the evolved controller: rapid movement toward the emitter, circling the emitter, and flying in a stable and efficient way.

3.4.1 Normalized distance

The primary goal of the UAV is to fly from its initial position to the radar site as quickly as possible. We measure how well controllers accomplish this task by averaging the squared distance between the UAV and the goal over all time steps. We normalize this distance using the initial distance between the radar and the UAV in order to mitigate the effect of varying distances from the random placement of radar sites. The normalized distance fitness measure is given as

$$fitness_1 = \frac{1}{T} \sum_{i=1}^T \left[\frac{d_i}{d_0} \right]^2 \quad (1)$$

where T is the total number of time steps, d_0 is the initial distance, and d_i is the distance at time i . We are trying to minimize $fitness_1$.

3.4.2 Circling distance

The secondary goal of the UAV is to circle closely around the source, since most applications of this type of controller require proximity to the target; when the UAV is within range of the target, it should circle around it. An arbitrary distance much larger than the desired circling radius is defined as the in-range distance. For this research, the in-range distance was set to be 10 nmi. The circling distance fitness metric measures the average distance between the UAV and the radar over the time the UAV is in range. The distance is squared to apply pressure to GP to evolve very small circling distances. While the circling distance is also measured by $fitness_1$, that metric is dominated by distances far away from the goal and applies very little evolutionary pressure to circling behavior. The circling distance fitness measure is given as

$$fitness_2 = \frac{1}{N} \sum_{i=1}^T inrange \cdot d_i^2 \quad (2)$$

where N is the amount of time the UAV spent within the in-range boundary of the radar and $inrange$ is 1 when the UAV is in-range and 0 otherwise. We are trying to minimize $fitness_2$.

3.4.3 Level time

In addition to the primary goals of moving toward a radar site and circling it closely, it is also desirable for the UAV to fly efficiently in order to minimize the flight time necessary to get close to the goal and to prevent potentially dangerous flight dynamics, like frequent and drastic changes in the roll angle. The first fitness metric that measures the efficiency of the flight path is the level time, the amount of time the UAV spends with a roll angle of zero degrees, which is the most stable flight position for a UAV. This fitness metric only applies when the UAV is outside the in-range distance; once the UAV is in range, we want it to circle around the radar, requiring a non-zero roll angle. The level time is given as

$$fitness_3 = \sum_{i=1}^T (1 - inrange) \cdot level \quad (3)$$

where *level* is 1 when the UAV has been level for two consecutive time steps and 0 otherwise. We are trying to maximize *fitness*₃.

3.4.4 Turn cost

The second fitness measure intended to produce an efficient flight path is a measure of turn cost. While UAVs are capable of quick, sharp turns, it is preferable to avoid them in favor of more gradual turns. The turn cost fitness measure is intended to penalize controllers that navigate using a large number of sharp, sudden turns because this behavior may cause unstable flight, even stalling. The UAV can achieve a small turning radius without penalty by changing the roll angle gradually; this fitness metric only accounts for cases where the roll angle has changed by more than 10° since the last time step. The turn cost is given as

$$fitness_4 = \frac{1}{T} \sum_{i=1}^T hardturn \cdot |\varphi_i - \varphi_{i-1}| \quad (4)$$

where φ is the roll angle of the UAV and *hardturn* is 1 if the roll angle has changed by more than 10° since the last time step and 0 otherwise. We are trying to minimize *fitness*₄.

3.5 Genetic Programming

We designed the four fitness functions to evolve particular behaviors, but the optimization of any one function could conflict heavily with the performance of the others. Combining the functions using multi-objective optimization is extremely attractive due to the use of non-dominated sorting. The population is sorted into ranks, where within a rank no individual is dominant in all four fitness metrics. Applying the term multi-objective optimization to this evolutionary process is a slight misnomer, because this research was concerned with the generation of behaviors, not optimization. In the same way that a traditional genetic algorithm can be used for both optimization and generation, so can multi-objective methods. Though this process isn't concerned with generating the most optimized controllers possible, it can obtain near-optimal solutions. In this research, we evolved UAV controllers using an implementation of NSGA-II (Deb et al., 2002) for GP. The multi-objective genetic algorithm employs non-dominated sorting, crowding distance assignment to each solution, and elitism.

The function and terminal sets used in this work combine a set of very common functions used in GP experiments with a set of functions specific to this problem. The function and terminal sets are defined as

$F = \{ Prog2, Prog3, IfThen, IfThenElse, And, Or, Not, <, \leq, >, \geq, <0, >0, =, +, -, *, \div, X<0, Y<0, X>max, Y>max, Amplitude>0, AmplitudeSlope>0, AmplitudeSlope<0, AoA>Arg, AoA<Arg \}$

$T = \{ HardLeft, HardRight, ShallowLeft, ShallowRight, WingsLevel, NoChange, rand, 0, 1 \}$

The UAV has a GPS on-board, and the position of the UAV is given by the *x* and *y* distances from the origin, located in the southwest corner of the simulation area. This position information is available using the functions that include *X* and *Y*, with *max* equal to 100 nmi, the length of one side of the simulation area. The UAV is free to move outside of this area during the simulation, but the radar is always placed within it. The two available sensor

measurements are the amplitude of the incoming radar signal and the AoA. Additionally, the slope of the amplitude with respect to time is available to GP. When turning, there are six available actions. Turns may be hard or shallow, with hard turns making a ten degree change in the roll angle and shallow turns a two degree change. The *WingsLevel* terminal sets the roll angle to 0, and the *NoChange* terminal keeps the roll angle the same. Multiple turning actions may be executed during one time step, since the roll angle is changed as a side effect of each terminal. The final roll angle after the navigation controller is finished executing is passed to the autopilot. The maximum roll angle is forty-five degrees. Each of the six terminals returns the current roll angle.

GP was generational, with crossover and mutation similar to those outlined by Koza (Koza, 1992). The parameters used by GP are shown in Table 1. Tournament selection was used. Initial trees were randomly generated using ramped half and half initialization.

Population size	500
Tournament size	2
Simulation runs per evaluation	30
Maximum initial GP tree depth	5
Maximum GP tree depth	21
Crossover rate	0.9
Mutation rate	0.05

Table 1. GP parameters

In GP, evaluating the fitness of the individuals within a population takes significant computational time. The evaluation of each individual requires multiple trials, 30 trials per evaluation in this research. During each trial, the UAV and the radar are placed randomly and four hours of flight time are simulated. Evaluating an entire population of 500 individuals for a single generation requires 15,000 trials. Therefore, using massively parallel computational processors to parallelize these evaluations is advantageous. In this research, the master-slave model of parallel processing was used. The data communication between master and slave processors was done using the Message Passing Interface (MPI) standard under the Linux operating system. The master node ran the GP algorithm and did all computations related to selection, crossover, and mutation. Evaluations of individuals in the population were sent to slave nodes. The parallel computer used for the experiments was a Beowulf cluster made up of 46 computers running Linux. Each computer had two 2.4 GHz Pentium 4 processors with hyper-threading, for a total of 92 processors in the cluster. Hyper-threading provides a small performance gain for multiple simultaneous processes, so two slave nodes were run on each processor, for a total of 184 slave nodes spread over the 92 processors in the cluster.

4. Evolution Experiments

We used multi-objective GP to evolve autonomous navigation controllers for UAVs. Controllers were evolved on radar types of varying difficulties. We evolved controllers using subsets of the four fitness functions in order to evaluate the effect of each fitness measure on controller behavior. In order to gauge the performance of evolution for multiple objectives, we devised test functions to measure the performance of a controller on the task. We then evolved controllers using both direct evolution and incremental evolution for five

radar types: continuously emitting, stationary radars; continuously emitting, mobile radars; intermittently emitting, stationary radars with regular emitting periods; intermittently emitting, stationary radars with irregular emitting periods; and intermittently emitting, mobile radars with regular emitting periods. In order to statistically measure the performance of GP on this problem, we did 50 evolutionary runs for each type of radar, where each run produced 500 controllers.

4.1 Effectiveness of Fitness Functions

To test the effectiveness of each of the four fitness measures, we evolved controllers using various subsets of the fitness metrics. These tests were done using the stationary, continuously emitting radar: this was the simplest of the radar types used for this research. The first fitness measure, the normalized distance, was included in every subset. The primary goal of the UAV is to fly from its initial position to the radar site as quickly as possible; $fitness_1$ is the only one of the four fitness functions that measures this behavior. When only $fitness_1$ was used to measure controller fitness, flight paths were very direct. The UAV flew to the target in what appeared to be a straight line. To achieve this direct route to the target, the controller would use sharp and alternating turns. The UAV would almost never fly level to the ground, and all turns were over 10° . Circling was also not consistent; the controllers frequently changed direction while within the in-range boundary of the radar, rather than orbiting in a circle around the target. For this simplest of fitness measures, evolution tended to select very simple bang-bang type control, changing the roll angle at every time step using sharp right and left turns.

Using only two fitness measures was not sufficient to achieve the desired behaviors. If $fitness_1$ and $fitness_2$ (circling distance) were used, the circling behavior improved, but the efficiency of the flight path was unchanged. If $fitness_1$ and $fitness_4$ (turn cost) were used, turns were shallower, but the UAV still failed to fly with its wings level to the ground for long periods. Circling around the target also became more erratic and the size of the orbits increased. If $fitness_1$ and $fitness_3$ (level time) were used, the UAV would fly level a large amount of the time, but circling was very poor, with larger radius orbits or erratic behavior close to the target. Sharp turns were also very common.

If three of the fitness measures were used, evolved behavior was improved, but not enough to satisfy the mission goals. If all fitness measures were used except $fitness_2$, the UAV would fly efficiently to the target, staying level and using only shallow turns. Once in range of the radar, circling was generally poor. Evolved controllers either displayed large, circular orbits or very erratic behavior that was unable to keep the UAV close to the radar. If $fitness_1$, $fitness_2$, and $fitness_4$ were used, the UAV would circle well once it flew in range of the radar. While flying toward the radar, the UAV failed to fly level, though turns tended to be shallow. The best combination of three fitness measures was when only $fitness_4$ was removed. In this case, circling was good and the UAV tended to fly straight to the target. The level time fitness measure also tended to keep the turns shallow and to eliminate alternating between right and left turns. However, turn cost was still high, as many turns were sharp.

When we used all four of the fitness functions, the evolved controllers were able to overcome a noisy environment and inaccurate sensor data in tracking and orbiting a radar site. A variety of navigation strategies were evolved to satisfy these fitness functions. All four fitness measures had an impact on the behavior of the evolved controllers, and all four were necessary to achieve the desired flight characteristics.

4.2 Evolution

In this research, we used both direct and incremental evolution. In direct evolution, controllers were evolved from random initial populations for continuously emitting, stationary radars; continuously emitting, mobile radars; intermittently emitting, stationary radars with regular periods; intermittently emitting, stationary radars with irregular periods; and intermittently emitting, mobile radars with regular periods. For each experiment, we performed 50 evolutionary runs and then selected successful controllers.

While all four objectives are important, moving the UAV to the goal is the highest priority. To emphasize this objective, controllers evolved directly from random initial populations used functional incremental evolution, which incrementally changes the fitness function to increase the difficulty of the problem. Only the normalized distance fitness measure was used for the first 200 generations; the last 400 generations used all four of the fitness functions.

To improve the chances of successfully evolving acceptable controllers for the more complex radar types, we used environmental incremental evolution. Unlike the direct evolution experiments, which always started with a random initial population, these experiments used evolved populations from simpler radar types as seed populations. Environmental incremental evolution incrementally increases the difficulty of the environment or task faced by evolution, while leaving the fitness function unchanged. In this research, random populations are initialized and then evolved for 600 generations on continuously emitting, stationary radars to create seed populations. Controllers for more difficult radars are then evolved for 400 generations using these seed populations. Populations were incrementally evolved on progressively more difficult radar types: continuously emitting, mobile radars; intermittently emitting, stationary radars; and intermittently emitting, mobile radars. Figure 2 graphically summarizes the process of incremental evolution used in this work.

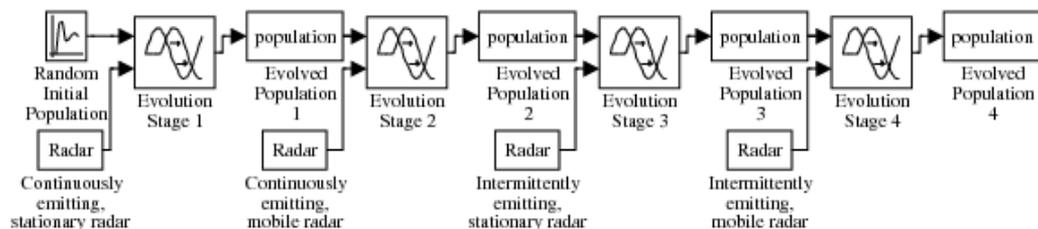


Figure 2. Environmental incremental evolution process

4.3 Test Metrics for Controller Evaluation

During controller evolution, four fitness functions determined the success of individual UAV navigation controllers. The fitness of a controller was measured over 30 simulation trials, where the UAV and radar positions were random for every trial. We designed the four fitness functions to measure how well a controller satisfied the goals of moving toward the radar, circling the radar closely, and flying in an efficient and stable manner.

These four fitness functions worked well to evolve good controllers, but because the functions were designed to exert evolutionary pressure throughout each run, not all the values each function produces are immediately meaningful. For the purposes of testing evolved controllers, we designed four test functions which measure the same qualities as the four fitness functions. The values these test functions produce are more meaningful to an observer.

4.3.1 Flying to the radar

The primary goal of the UAV is to fly from its initial position to the radar site as quickly as possible. The first fitness function, $fitness_1$, measured how well controllers accomplish this task by averaging the squared distance between the UAV and the goal over all time steps. We normalized this distance using the initial distance between the radar and the UAV in order to mitigate the effect of varying distances from the random placement of radar sites. However, this measure does include a slight bias against longer initial distances, and produces a value without much meaning for an observer. We eliminated this bias in the first test function, $test_1$, by measuring percent error in flight time to the target. The total simulated time of four hours, or 14400 seconds, is divided into T_{in} , the number of seconds the distance between the UAV and radar is less than 10 nmi, and T_{out} , when this distance is greater than or equal to 10 nmi.

$$T_{total} = T_{in} + T_{out} = 14400 \text{ seconds} \quad (5)$$

The error in the time it takes to fly to the radar is just the actual time, T_{out} , minus the shortest possible time, T_{expect} , which is computed from D , the shortest possible distance in nautical miles a UAV could travel to fly from its starting position to each radar location, and the UAV speed of 80 knots.

$$T_{expect} = \frac{D}{80 \frac{nmi}{hour} \cdot \frac{1 \text{ hour}}{3600 \text{ seconds}}} = 45 \cdot D \quad (6)$$

The test function is given as

$$test_1 = \left[\frac{T_{out} - T_{expect}}{T_{expect}} \right] \quad (7)$$

For our tests, a value for $test_1$ near zero indicates a good flight.

4.3.2 Circling the radar

In early tests, we found that finding the mean squared circling distance exerted more pressure to evolve good circling behavior than if we simply used the mean circling distance. The circling distance fitness function used to evolve the controllers used the mean squared distance between the UAV and the radar when this distance was less than 10 nmi. For our tests, we were more concerned with the actual mean circling distance, so the test function, $test_2$, is the mean circling distance between the UAV and the radar when this distance is less than 10 nmi. The circling distance test function is

$$test_2 = \frac{1}{T_{in}} \sum_{i=1}^T inrange \cdot d_i \quad (8)$$

where $inrange$ equals 1 if the distance between the UAV and the radar is less than 10 nmi and 0 otherwise.

4.3.3 Efficient flight

The first fitness function used to measure the efficiency of flight, $fitness_3$, is the number of time steps the UAV spends with a roll angle of 0° while traveling to the target. When the mean value of this fitness function is taken over many simulated flights, it provides a good measure of the amount of time a UAV spends flying in the most efficient posture. For the ability to look at single flights as well as a large number of simulations, we created $test_3$, which measures the percentage of the expected time the UAV spends flying level.

$$test_3 = \frac{1}{T_{\text{expect}}} \left| \left(\sum_{i=1}^T (1 - inrange) \cdot level \right) - T_{\text{expect}} \right| \quad (9)$$

where $level$ is 1 when the UAV has been level for two consecutive time steps and 0 otherwise. For our tests we would like $test_3$ to be as small as possible.

4.3.4 Stable flight

The second test function to evaluate the efficiency of flight is a measure of turn cost. While UAVs are capable of quick, sharp turns, it is preferable to avoid these in favor of more gradual turns. The original turn cost fitness function $fitness_4$, also used as $test_4$, was intended to penalize controllers that navigate using a large number of sharp, sudden turns because this behavior may cause unstable flight or stalling. The UAV can achieve a small turning radius without penalty by changing the roll angle gradually; the metric only accounts for cases where the roll angle has changed by more than 10° since the last time step. The turn cost is given as

$$test_4 = \frac{1}{T} \sum_{i=1}^T hardturn \cdot |\varphi_i - \varphi_{i-1}| \quad (10)$$

where φ is the roll angle of the UAV and $hardturn$ is 1 if the roll angle has changed by more than 10° since the last time step and 0 otherwise. We would like to minimize $test_4$.

4.4 Controller Evaluation

Since multi-objective optimization produces a Pareto front of solutions, rather than a single best solution, we needed a method to gauge the performance of evolution. To do this, we selected values we considered acceptable for the four fitness metrics. We defined a minimally successful UAV controller as able to move quickly to the target radar site, circle at an average distance under 2 nmi, fly with a roll angle of 0° for approximately half the distance to the radar, and turn sharply less than 0.5% of the total flight time. If a controller had $test_1$ less than 0.2, $test_2$ less than 2, $test_3$ less than 0.5, and $test_4$ less than 0.05, the evolution was considered successful. These baseline values were used only for our analysis, not for the evolutionary process.

4.5 Direct Evolution

Table 2 shows the number of successful runs and the success rate for each of the five radar types and the total number of successful controllers, average number of successful

controllers for an evolutionary run, and maximum number of controllers evolved in an evolutionary run for each of the radar types using direct evolution.

Radar type	Evolutionary runs			Successful controllers		
	Total	Successful	Percent	Total	Average	Maximum
Continuous, stationary	50	45	90%	3,149	62.98	170
Continuous, mobile	50	36	72%	2,266	45.32	206
Inter. (regular), stationary	50	25	50%	1,891	37.82	156
Inter. (irregular), stationary	50	29	58%	2,374	47.48	172
Intermittent, mobile	50	16	32%	569	11.38	93

Table 2. Number of successful runs and controllers for direct evolution experiments

Unlike continuously emitting radars, intermittently emitting radars were quite difficult for evolution. This should come as no surprise; the sensors on-board the UAV receive only half as much information from this type of radar as from a continuously emitting radar. Since the controllers evolved in this research have no *a priori* knowledge of the radar location and no internal model of the world, evolution must devise a strategy for times when the emitter is turned off. Despite the increased difficulty of this experiment, evolution was able to produce a large number of successful controllers.

4.6 Incremental Evolution

The results of the incremental evolution experiments are shown in Table 3.

Radar type	Evolutionary runs			Successful controllers		
	Total	Successful	Percent	Total	Average	Maximum
Continuous, stationary	50	45	90%	2,815	56.30	166
Continuous, mobile	50	45	90%	2,774	55.48	179
Intermittent, stationary	50	42	84%	2,083	41.66	143
Intermittent, mobile	50	37	74%	1,602	32.04	143

Table 3. Number of successful runs and controllers for incremental evolution experiments

To begin the incremental evolution process, we evolved controllers for continuously emitting, stationary radars. This new set of evolutionary runs was used as a seed for the incremental evolution experiments. Like the experiments described in Section 4.5, 45 of the 50 evolutionary runs were successful, for a success rate of 90%

In the second stage of incremental evolution, each of the seed populations was used as the initial population for an evolutionary run, which evolved for 400 generations on continuously emitting, mobile radars. The use of incremental evolution improved the success rate of evolution on this type of radar. The 90% success rate using incremental evolution was an increase over the 72% success rate using direct evolution.

In the third stage of incremental evolution, each of the populations evolved in the second stage was used as a seed population for 400 generations of evolution on intermittently emitting, stationary radars with regular periods. The use of multiple increments, or stages of evolution, dramatically increased the ability of evolution to produce adept controllers for this type of radar. The success rate for evolution on intermittently emitting, stationary radars increased from 50% for direct evolution to 84% in this experiment. This increase in

success rate suggests that incremental evolution is a very effective technique for this problem.

In the fourth and final stage of incremental evolution, each of the populations evolved the third stage was used as a seed population to evolve controllers for intermittently emitting, mobile radars with regular periods over 400 generations. Using multiple stages of incremental evolution increased the ability of evolution to successfully produce good results for this radar type. The success rate for intermittently emitting, mobile radars was 32% for direct evolution, but the success rate jumped all the way to 74% with incremental evolution.

4.7 Transference to a Wheeled Mobile Robot

To evaluate the ability of evolved controllers to control real vehicles, we transferred evolved UAV navigation controllers to a wheeled mobile robot. We used a small autonomous mobile robot called the EvBot II (Mattos, 2003), shown in Figure 3. The robot is equipped with an on-board computer responsible for all computation, data acquisition and high-level control. The robot is connected to a wireless network and supports video data acquisition through a USB video camera. The EvBot is equipped with an on-board passive sonar system that makes use of an acoustic array formed by eight microphones distributed in a fixed 3-D arrangement around the robot. It uses data collected from the array to perform beamforming and to find the direction and intensity of sound sources. The passive sonar system is susceptible to environmental noise, and the direction of a source found by the acoustic array is only accurate within approximately $\pm 45^\circ$.

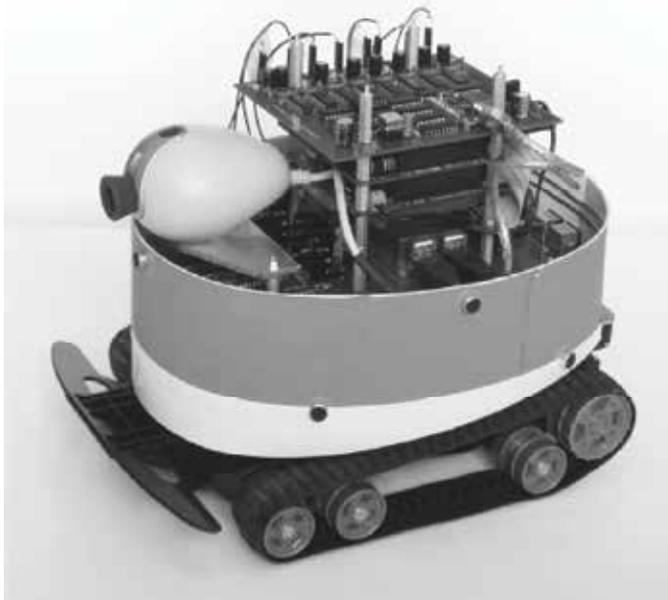


Figure 3. The EvBot II mobile robot equipped with an acoustic array

Testing the evolved controllers on the EvBot was attractive because the passive sonar system is an acoustic analog to the radar sensor used in simulation. The two signal types propagate similarly, and both sensors detect signal strength and direction. The similarities between the two systems made it possible to transfer evolved UAV controllers to an EvBot. In evaluating

the transference of the evolved controllers, we were not interested in showing optimal behavior on the robot platform. Instead, our concern was that the controllers should exhibit the same behaviors on the real robots as they did in simulation, particularly robustness to noise. Since the sensor accuracy of the acoustic array was so much worse than that of the AoA sensor in our simulation, we evolved new controllers to transfer to the EvBot. The only change in the simulation was changing the AoA accuracy from $\pm 10^\circ$ to $\pm 45^\circ$.

Transference experiments were done in a 153 inch by 122 inch arena. A video camera with a fisheye lens was mounted above the maze environment to document experiments. In each experiment, the robot was placed along one wall facing toward the middle of the environment. A speaker was suspended a foot above the ground and continuously emitted a 300 Hz tone. A circle was placed directly underneath the speaker as a visual reference point, since the fisheye lens tended to distort the location of the speaker in images captured by the overhead camera. Robot movement was discretized into steps, much like in the simulation. At each time step, the controller was executed to produce a roll angle. The EvBot was only calibrated to turn at multiples of 5° : calibrating the EvBot to turn at angles smaller than 5° would have been unreliable due to the size of the EvBot and the characteristics of its motors. After turning, the EvBot would always move forward the same amount, mimicking the constant speed of the UAV in simulation. The EvBot moved 3 inches per time step, and in simulation the UAV moved 0.02 nautical miles per time step. If these values are used to scale the maze environment, then the maze would represent an area approximately 1.13 nmi by 0.90 nmi. Hence, these experiments were not testing the entire flight path, only the very end of flight when the vehicle nears the target.

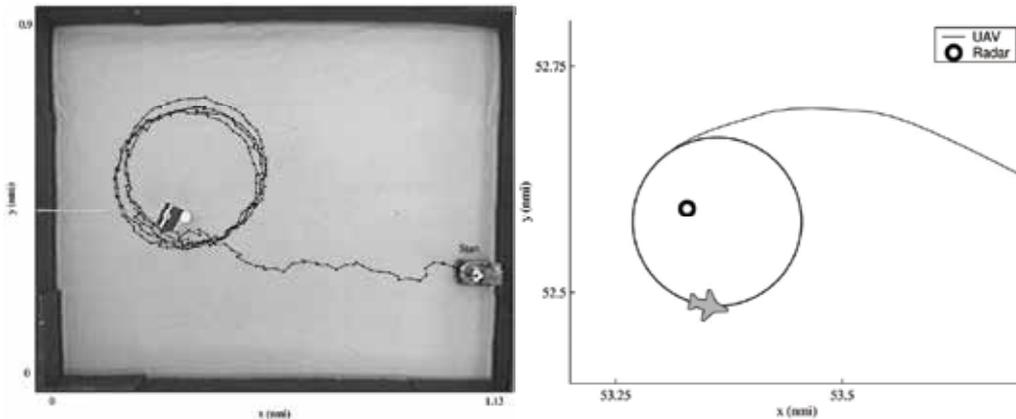


Figure 4. Circling behavior comparison for the EvBot (scaled maze size shown in nmi) and a simulated UAV, both running evolved controllers

Controllers evolved with a less accurate sensor were not as well adapted as those from previous work using more accurate sensors; flight paths were much less smooth and required more turns (Barlow et al., 2005). An evolved controller was tested 10 times on an EvBot. We chose this controller from the evolved population based primarily on good fitness values for normalized distance and circling distance, though level time and turn cost were also used. This controller was able to successfully drive the EvBot from its starting position to the speaker and then circle around the speaker. This small number of tests was enough to confirm that the controllers were consistently able to perform the task as desired.

Figure 4 shows a path from one of the experiments compared to the circling behavior in simulation of controllers evolved with $\pm 10^\circ$ sensor accuracy. Running this evolved controller on the EvBot produces a tight circling behavior with a regular orbit around the target.

5. Robustness Analysis of GP Navigation Controllers

Over the 50 evolutionary runs with populations of 500 for each run, we produced 25,000 GP trees. In each evolutionary run, all 500 members of the final population fell along the Pareto front. Many of these controllers, however, only performed well according to one of the test metrics described in Section 4.3 while doing poorly at the others. The evolved controller best suited for transference to a real UAV would perform well on all of the test metrics. Rather than thoroughly testing all of the evolved controllers, including the most poorly performing, we chose to test only a small number of the best controllers. We established several performance metrics to evaluate controllers. Through successive performance metric evaluations, we selected the 10 best controllers for testing and subjected these evolved controllers to a series of robustness tests.

5.1 Performance Metrics

Multi-objective optimization produces a Pareto front of solutions, rather than a single best solution. In order to rank the controllers, each performance metric should combine the four test functions into a single value. The basis for all the performance metrics are a set of baseline values, values for each test function that describe a minimally successful UAV controller, defined in Section 4.4. Controllers are compared using four performance metrics: 1) *failures*, 2) *normalized maximum*, 3) *normalized mean*, and 4) *average rank*.

The first performance metric, *failures*, measures the percentage of flights with test function values which fail to meet at least one of the baseline values. A simulation run is a failure if there exists an m such that $test_m(r,n)$ is greater than $baseline_m$, where $test_{1...4}(r,n)$ are the values of the four test functions for simulation run n for radar r . The failure percentage for a controller f and a test t is given as

$$metric_1(f,t) = \frac{F}{N} \quad (11)$$

where N is the total number of simulations and F is the number of simulation runs that fail. The second performance metric, *normalized maximum*, measures how poorly a controller does when it fails. While the *failures* performance metric measures how often a controller fails, it does not measure how badly it might fail. Some controllers might perform well most of the time, but do not fail gracefully. The *normalized maximum* performance metric measures the worst failure for a particular controller. For each test function, the largest of the N values for each R radar is normalized by the baseline value for that function. Each value $test_m(r,n)$ is described by the test function (m), the radar type (r), and the simulation number (n). The maximum value over the M test functions is the *normalized maximum*, given as

$$metric_2(f,t) = \max_M \left(\frac{\max_{R,N} (test_m(r,n) - baseline_m)}{baseline_m} \right) \quad (12)$$

The third performance metric, *normalized mean*, measures how well a controller performs in relation to the baseline values. While the two metrics above measure the consistency of the controller and how wildly it can fail, this metric shows the typical performance of a controller. The *normalized mean* is given as

$$metric_3(f, t) = \frac{1}{M} \sum_{m=1}^M \left(\frac{baseline_m - \frac{1}{R} \sum_{r=1}^R \frac{1}{N} \sum_{n=1}^N test_m(r, n)}{baseline_m} \right) \quad (13)$$

The test function values for each objective are first averaged over the number of samples N , then over the number of radars R . For each objective, this average is normalized by the corresponding baseline value. We compute the normalized mean by taking the average over the M objectives.

The fourth performance metric, *average rank*, combines the values from the first three metrics into a single metric. To measure the relative performance of the controllers and give each metric equal weight, the values for all g controllers are normalized to be between 0 and 1.

$$norm_k(f, t) = \frac{metric_k(f, t) - \min_G(metric_k(g, t))}{\max_G(metric_k(g, t)) - \min_G(metric_k(g, t))} \quad (14)$$

The value of $metric_4(f, t)$ is the mean of these normalized metrics.

$$metric_4(f, t) = \frac{1}{3} \sum_{k=1}^3 norm_k(f, t) \quad (15)$$

If we wish to find $metric_4(f, T)$ where T is a set of tests, we find $metric_k(f, T)$ values for each of the first three metrics

$$metric_k(f, T) = \frac{1}{T} \sum_{t=1}^{|T|} metric_k(f, t) \quad (16)$$

then normalize using Equation 14 and compute the metric using Equation 15.

5.2 Controller Selection

The combination of simulated sensor noise and random positioning of UAVs and radars created an uncertain fitness landscape for this problem. During evolution, we averaged the values from 30 simulation trials to help mitigate this uncertainty, but for these robustness tests, orders of magnitude more tests would make test function values for individual controllers statistically meaningful. Rather than running thousands of simulations for each of the 25,000 controllers created by evolution—an approach that would have been too computationally expensive—we chose to perform a series of robustness tests on 10 of the best controllers. We selected these controllers over several stages, reducing the number of controllers by an order of magnitude during each step.

First, we selected all controllers whose mean was lower than the baseline values for the four test functions described in Section 4. Of the 25,000 evolved controllers, 1,602 controllers had average fitness values better than the baseline values. This first method of selection was chosen because these 1,602 controllers had already been shown to perform well on the five radar types of interest: continuously emitting, stationary radars; continuously emitting, mobile radars; intermittently emitting, stationary radars with regular emitting periods; intermittently emitting, stationary radars with irregular emitting periods; and intermittently emitting, mobile radars with regular emitting periods.

Second, we ran 100 simulation trials on each of the five radar types for each of the 1,602 controllers and measured each flight using the test functions outlined in Section 4.3. For each radar type, we selected the best 35% of controllers using the *failures* performance metric described above. Then, we took the intersection of these five sets of controllers, leaving 298 controllers out of the 1,602. This selection method was chosen to eliminate controllers that did not perform consistently well on all five of the radar types. Since some radar types were more difficult than others, using a single cutoff number of failures to select controllers would not have caught controllers that did not perform as well on the easier radar types. The choice of 35% was made in order to select approximately 300 controllers for further tests.

Finally, we cut these 298 controllers down to 10 using the *normalized maximum* performance metric. This performance metric was applied to all 298 controllers and the 10 controllers with the lowest *normalized maximum* were selected as the best controllers for further testing. While counting the number of times a controller fails to meet the baseline values is a good way to compare controllers, this method gives no indication of the magnitude of failure. Using the *normalized maximum* metric helped eliminate controllers that usually performed well, but occasionally performed extremely poorly. Since a consistently sub-optimal controller is preferable to an unpredictable one, this metric was a good way to cut the set of controllers to a small number for final testing.

We compared these 10 evolved controllers to two designed controllers, a hand-written controller specific to this domain and a proportional-derivative (PD) controller, a common feedback controller. We did not use a proportional-integral-derivative (PID) controller because intermittent and mobile radars made the integral term detrimental to performance in preliminary tests. After examining many successful evolved controllers, we designed the hand-written controller using only the function set available to GP. We used strategies seen in evolved controllers, but tried to reduce the complexity to an easy to understand controller that performed well under the same conditions used in evolution. Given the current AoA, amplitude, and roll angle as inputs, if the amplitude is greater than zero, the hand-written controller will make a turn of fixed magnitude if necessary. If the AoA is greater than 10° , the roll angle will be increased. If the AoA is less than -10° , the roll angle will be decreased. If the AoA is between 10° and -10° , and the magnitude of the roll angle is greater than zero, the roll angle will be increased or decreased to move it closer to zero. Otherwise, the roll angle remains at 0° . The PD controller takes as input the current AoA and the AoA at the previous time step. The derivative of AoA is approximated by using the difference between the AoA at the previous time step and the current AoA. We adjusted the proportional and derivative gains to give good performance under the same conditions used for evolution.

5.3 Robustness Testing

During evolution, controller evaluation simulated an aircraft with constant speed, noise on the two sensor measurements (angle of arrival (AoA) and amplitude), and no state noise. The airspeed was 80 knots, AoA noise during evolution was $\pm 10^\circ$, and the amplitude noise was $\pm 6\text{dB}$. To test the robustness of the evolved controllers, we increased the sensor noise and introduced sources of state noise. In addition to a control case with conditions identical to those during evolution, robustness tests fell into five categories by the type of noise: 1) AoA error, 2) amplitude error, 3) UAV airspeeds different from the speed used in evolution, 4) heading error, 5) and wind effects (position error). For each robustness test, we tested ten evolved controllers, a hand-written controller, and a PD controller against all five radar types. For every combination of controller and radar, we performed 10,000 simulation runs, for a total of 50,000 simulations for each controller per robustness test.

The sensor used most by evolved controllers was the AoA sensor. To test the robustness of evolved controllers, we varied the accuracy of the AoA sensor. The apparent AoA is given as the true angle to the target plus a normally distributed random number times the AoA accuracy. This accuracy was set to $\pm 10^\circ$ during evolution. For these robustness tests, we used AoA accuracies of $\pm\{15^\circ, 20^\circ, 25^\circ, 30^\circ\}$.

While the amplitude sensor was not often used by evolved controllers, we did one test where we increased the amplitude error. While the controllers were being evolved, the amplitude error was set to 6 dB; the robustness test used an error of 12 dB. In both cases, this error was multiplied by a normally distributed random number and added to the true amplitude to compute the apparent amplitude.

During evolution, the speed of the UAV was held constant at 80 knots, but a UAV can obviously be flown at a variety of speeds. For instance, the UAV being considered for flight tests has a stall speed of 40 knots and a top speed of 110 knots. To test the performance of the evolved controllers at different speeds, we chose to test at 50 and 100 knots, values near the low and high end of the speed range.

The evolved UAV controllers have a single control variable, roll angle, which is used by the simulation to change the heading of the UAV at each time step. In the version of the simulation used to evolve controllers, there was no noise in this process. It is possible that on a real UAV, the autopilot might not be able to respond perfectly to turn requests or wind might push the UAV off-course. To test the robustness of evolved controllers this possibility, we added noise to the heading state variable. At each time step, a normally distributed random variable multiplied by a heading error was added to the heading computed using the desired roll angle. For our tests, we used heading error values of $\pm\{0.5^\circ, 1.0^\circ, 1.5^\circ, 2.0^\circ\}$.

A significant source of state error for UAVs in the real world is wind. During evolution, the effects of wind on a UAV were not taken into account. While wind may function as a source of heading noise, the largest effect examined here was on position error. For our robustness tests, we make the simplifying assumption that the wind acts as an external force vector that can be summed with the propulsion force vector to obtain the new position of the UAV. In reality, this assumption is pessimistic, but for the purposes of these robustness tests would help to gauge the effects of wind on evolved controllers. The wind direction was set randomly for each simulation, but once set, the wind direction was held constant for the duration of the simulation. The wind speed was calculated at each time step as the mean wind speed plus some variance. For our tests, we used wind speeds of $\{5, 10, 20, 30\}$ knots with a variances of $\{1, 1, 5, 5\}$ knots.

5.4 Robustness Test Results

For each test, we ranked the 12 controllers (10 evolved controllers, labeled A to J; the hand-written controller, hd; and the PD controller, pd) based on each of the four performance metrics. Most of the results presented are rankings over several tests. When ranking over several tests, we used the averages of the performance metric values for each test except for the *average rank* metric; the technique used to compute this metric is described in Section 5.1.

Overall Ranking	best	2	3	4	5	6	7	8	9	10	11	12
failures	G	D	E	F	J	H	A	C	B	pd	I	hd
normalized maximum	pd	D	I	F	G	hd	J	E	B	A	H	C
normalized mean	D	E	hd	G	F	J	H	pd	A	C	B	I
average rank	D	G	E	pd	F	J	H	hd	A	B	C	I
Initial Ranking	best	2	3	4	5	6	7	8	9	10	11	12
Failures	pd	A	B	C	D	E	F	G	H	I	J	hd
normalized maximum	pd	G	E	J	F	I	B	A	D	C	H	hd
normalized mean	pd	I	J	A	B	C	G	E	D	F	H	hd
average rank	pd	J	G	I	E	B	A	F	C	D	H	hd
AoA Ranking	best	2	3	4	5	6	7	8	9	10	11	12
Failures	G	D	E	F	I	H	J	B	A	C	pd	hd
normalized maximum	pd	D	I	E	G	hd	B	J	H	F	A	C
normalized mean	D	G	E	hd	F	J	H	I	C	B	A	pd
average rank	G	D	E	F	pd	J	hd	H	I	B	C	A
Heading Ranking	best	2	3	4	5	6	7	8	9	10	11	12
Failures	D	E	F	G	J	H	pd	A	C	B	I	hd
normalized maximum	pd	I	D	J	H	F	B	hd	A	G	E	C
normalized mean	pd	D	H	J	F	hd	E	I	C	G	B	A
average rank	pd	D	J	F	H	E	G	I	B	C	A	hd
Wind Ranking	best	2	3	4	5	6	7	8	9	10	11	12
Failures	D	E	G	pd	F	J	H	C	B	A	hd	I
normalized maximum	pd	G	F	E	D	hd	I	J	A	B	C	H
normalized mean	pd	hd	D	G	E	F	J	H	A	C	B	I
average rank	pd	G	D	E	F	J	H	hd	A	C	B	I

Table 4. Controller rankings for robustness tests

Controller rankings, as shown in Table 4, are divided into five separate sections. The first section shows the overall rankings, averaged over all 16 robustness tests described in Section 5.3. The second section shows the initial rankings, using the same conditions under which the controllers were evolved (the control test). The third section shows the AoA rankings, averaged over the control test and the four tests with decreased AoA accuracy. The results from the next three tests, increasing the amplitude noise and changing the UAV speed to 50 and 100 knots, were very similar to those for the control test, so these results are not shown in the interests of space. The fourth section shows the heading rankings, averaged over the control test and the four tests with increased heading error. The fifth section shows the wind rankings, averaged over the control test and the four tests with increased wind speed. Based on these rankings, evolved controller D was the best candidate to transfer to a real UAV. In all rankings, controller D, the hand-written controller, and the PD controller are highlighted. The failure percentages for controller D and the PD controller for each of the robustness tests for each radar type are shown in Table 5.

In the initial ranking, the PD controller performs better than all the evolved controllers and the hand-written controller on all four metrics. This was not surprising, as we had tuned the controller parameters for the control test, giving an average failure rate of 0.04%. The hand-written controller, which was not optimized, performed the worst with an average failure rate of 69.82%. Controller D performed well, with an average failure rate of 7.13%, but was ranked tenth on the *average rank* metric, ahead of only one other evolved controller and the hand-written controller.

As we increased AoA noise, the performance of the hand-designed controllers declined compared to the evolved controllers. When the AoA error was increased from $\pm 10^\circ$ to $\pm 15^\circ$, the PD controller failed in 100% of tests and the hand-written controller failed in 92.72% of tests. On the other hand, the failure rate for controller D was only 9.88%. The average failure rate for controller D only rose above 25% (to 90.24%) once the AoA error was $\pm 30^\circ$. For the five different settings of AoA accuracy, controller D was the most robust to AoA sensor noise of the controllers.

An increase in amplitude error did not significantly change the performance of any controller. The performances of the evolved controllers and the hand-written controller on the two tests varying the speed were similar to performances on the control case. The only controller that had trouble with different speeds was the PD controller, which had average failure rates of 100% for a speed of 50 knots and 92.69% for a speed of 100 knots. In addition to being tuned for a particular AoA accuracy, these tests suggest that the PD controller is also tuned for a particular airspeed.

For the robustness tests with heading noise, the PD controller was the best, followed by controller D, which was the most consistent of the evolved controllers. The hand-written controller was the worst of the 12 controllers over these four tests and the control. Controller D actually failed significantly less than the PD controller; the average failure rate for the PD controller was over 50% when the heading error was $\pm 1.5^\circ$ and was 98.6% when the error was $\pm 2^\circ$, while the average failure rate for controller D never got above 20%. The PD controller typically did not fail by large margins, and was ranked first on the *normalized maximum* and *normalized mean* performance metrics.

In the last series of tests, we added a different source of state error, wind. This series of tests clearly separated the evolved controllers; some of these controllers were simply not robust to the effects of wind. Controller D failed the fewest times, but was third in the *average rank*

metric. The PD controller was best in this category. Despite the large number of evaluations, there is still some uncertainty in the performance metric values—for example, when AoA error is increased from the control test, the failure rate for controller D on continuous, mobile radars actually decreases slightly. For most of the robustness tests, this uncertainty was small—in the previous example this drop was on the order of 0.5%—but for wind this uncertainty was more apparent, especially for the higher wind speeds. One artifact of this uncertainty was the change in the failure rate for controller D on continuous, stationary radars from wind speeds of 10 to 20 to 30 knots. At 10 knots, the failure percentage was 16.33%. When the wind speed was increased to 20 knots, the failure percentage increased to 78.96%. However, when the wind speed was increased again to 30 knots, the failure percentage dropped to 61.05%. Other evolved controllers showed similar trends for increased wind speeds.

Test type	Cs		cm		irs		iis		irm		average	
	D	Pd	D	pd								
control	0.00	0.04	10.19	0.03	1.18	0.07	10.39	0.04	13.87	0.03	7.13	0.04
AoA=15	0.00	100.0	9.67	100.0	6.55	100.0	16.43	100.0	16.74	100.0	9.88	100.0
AoA=20	0.00	100.0	9.80	100.0	20.10	100.0	26.06	100.0	24.81	100.0	16.15	100.0
AoA=25	0.01	100.0	9.40	100.0	35.39	100.0	37.76	100.0	34.90	100.0	23.49	100.0
AoA=30	99.65	100.0	99.05	100.0	77.15	100.0	90.12	100.0	85.25	100.0	90.24	100.0
Amp=12	0.00	0.04	10.83	0.02	1.41	0.04	10.73	0.04	13.21	0.07	7.24	0.04
Speed=50	0.00	100.0	12.38	100.0	0.56	100.0	3.70	100.0	16.53	100.0	6.63	100.0
Speed=100	0.00	92.92	10.28	92.42	2.63	92.23	17.04	92.79	14.83	93.09	8.96	92.69
Head=0.5	0.00	0.15	9.76	0.20	1.70	0.19	11.70	0.15	14.38	0.18	7.51	0.17
Head=1.0	0.00	2.57	10.84	2.46	4.48	2.50	16.88	2.66	16.08	2.60	9.66	2.56
Head=1.5	0.00	54.97	11.16	55.64	9.29	54.30	25.86	55.36	20.31	55.03	13.32	55.06
Head=2.0	0.00	98.56	11.04	98.57	19.85	98.55	37.63	98.67	29.38	98.64	19.58	98.60
Wind=5	0.02	0.39	11.43	0.00	2.40	0.01	12.52	0.05	15.70	0.14	8.41	0.12
Wind=10	16.33	0.05	25.66	0.00	21.13	1.39	30.97	0.02	30.82	1.11	24.98	0.51
Wind=20	78.96	97.54	72.10	94.76	49.96	95.12	72.05	100.0	60.77	94.83	66.77	96.45
Wind=30	61.05	98.36	71.20	100.0	63.43	96.64	95.42	97.28	86.10	99.67	75.44	98.39

Table 5. Failure percentages for the best evolved controller (D) and the PD controller (pd) listed by radar type (cs: continuous, stationary; cm: continuous, mobile; irs: intermittent, regular period, stationary; iis: intermittent, irregular period, stationary; irm: intermittent, regular period, mobile; avg: average)

The PD controller, as one might expect, performed extremely well under design conditions. When measuring how badly it failed using the *normalized maximum* performance metric, it was robust to all sources of noise, outperforming the other controllers in all tests. This

controller was also robust to state noise, performing well under heading error and wind. However, the PD controller was susceptible to sensor noise. As the AoA error increased a small amount, the PD controller quickly failed. It was also dependent on the speed of the aircraft for good performance.

On most tests, the hand-written controller performed well, but was consistently worse than evolved controllers. This controller typically produced good results on the *normalized mean* performance metric, but was not robust to most forms of noise, though it did perform well on the speed and wind tests. The hand-written controller was included in these tests to show the difficulty of using the function and test sets to design an optimal controller by hand.

Overall, the best and most consistent controller was the best of our evolved controllers, controller D. In the overall rankings, this controller had the best *average rank* and finished in the top two on every performance metric. Unlike the hand-designed controllers, there was no category of tests where controller D performed poorly. This controller ranked highly in all of our tests, and its rank tended to increase as noise was increased.

6. Conclusions

Using multi-objective GP, we were able to evolve navigation controllers for UAVs capable of flying to a target radar, circling the radar site, and maintaining an efficient flight path, all while using inaccurate sensors in a noisy environment. Controllers were evolved for five radar types using both direct evolution and incremental evolution: continuously emitting, stationary radars; continuously emitting, mobile radars; intermittently emitting, stationary radars with regular periods; intermittently emitting, stationary radars with irregular periods; and intermittently emitting, mobile radars with regular periods. The use of incremental evolution dramatically increased the chances of producing successful controllers compared to direct evolution. Incremental evolution also produced controllers able to handle all five radar types.

Controllers were evolved to use inaccurate sensors in a noisy environment. We tested the transferability of the evolved controllers by using them to control a wheeled mobile robot. Evolved UAV controllers were successfully transferred to a wheeled mobile robot equipped with a passive sonar system which provided the angle and amplitude of sound signals from a stationary speaker. Using evolved navigation controllers, the mobile robot moved to the speaker and circled around it. The results from this experiment demonstrate that our evolved controllers are capable of transference to real physical vehicles.

We developed a series of robustness tests for evolved navigation controllers for UAV controllers developed in simulation. Before testing evolved controllers on a real UAV, we needed some assurance that the off-design performance of these controllers would be sufficient to accomplish the desired task and that controllers would be able to avoid behaviors that could potentially damage the aircraft. Also, since the controllers were evolved using multi-objective optimization, we needed to select a single best controller. When evolving controllers for systems where tests may be dangerous to the vehicle, robustness tests might be useful in selecting a controller and seeing how well it performs. The robustness tests described here apply several sources of sensor and state noise. If the real-world noise falls within the range where tests in simulation performed well, we can expect that transference will be successful.

The best evolved controller performed well in all robustness tests, consistently outperforming a hand-written controller and a proportional-derivative controller. When subjected to reasonable noise, this controller continued to perform well, even when many other controllers began to fail. Despite outperforming the hand-designed controllers and the other evolved controllers, our best controller has limits. When the AoA sensor was very inaccurate or the wind speed was high, this controller did not perform very well.

We have described the systematic development of a UAV navigation controller. We designed fitness functions and evolved controllers with multi-objective GP, using incremental evolution to increase the number of good controllers. To evaluate the ability of evolved controllers to control real vehicles, we transferred evolved UAV navigation controllers to a wheeled mobile robot. Finally, we used a series of robustness tests to screen the controllers and to select the single best controller. Based on this research, we feel confident that GP can evolve autonomous navigation controllers to effectively and safely control real UAVs in real flight environments.

7. References

- Barlow, G.J. (2004). Design of autonomous navigation controllers for unmanned aerial vehicles using multi-objective genetic programming. *Master's thesis*, North Carolina State University, Raleigh, North Carolina.
- Barlow, G.J.; Mattos, L.S.; Oh, C.K. & Grant, E. (2005). Transference of evolved unmanned aerial vehicle controllers to a wheeled mobile robot, *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2087-2092, Barcelona, Spain, April 2005.
- Barlow, G.J. & Oh, C.K. (2006). Robustness Analysis of Genetic Programming Controllers for Unmanned Aerial Vehicles, *Proceedings of the Genetic And Evolutionary Computation Conference*, pp. 135-142, Seattle, Washington, July 2006.
- Barlow, G.J.; Oh, C.K. & Grant, E. (2004). Incremental evolution of autonomous controllers for unmanned aerial vehicles using multi-objective genetic programming, *Proceedings of the IEEE Conference on Cybernetics and Intelligent Systems*, pp. 688-693, Singapore, December 2004.
- Deb, K.; Agrawal, S.; Pratap, A. & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 2, 182-197.
- Ebner, M. (1998). Evolution of a control architecture for a mobile robot, *Proceedings of the Second International Conference on Evolvable Systems*, pp. 303-310, Lausanne, Switzerland, September 1998.
- Filliat, D.; Kodjabachian, J. & Meyer, J.-A. (1999). Incremental evolution of neural controllers for navigation in a 6-legged robot, *Proceedings of the International Symposium on Artificial Life and Robots*, Oita, Japan, January 1999.
- Gomez, F. & Miikkulainen, R. (1997). Incremental evolution of complex general behavior. *Adaptive Behavior*, Vol. 5, No. 3-4, 317-342.
- Gomez, F. & Miikkulainen, R. (2004). Transfer of neuroevolved controllers in unstable domains, *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 957-968, Seattle, Washington, June 2004.

- Harvey, I.; Husbands, P. & Cliff, D. (1994). Seeing the light: Artificial evolution, real vision, *Proceedings of the International Conference on Simulation of Adaptive Behavior*, pp. 704-720, Brighton, UK, August 1994.
- Hoffmann, F.; Koo, T.J. & Shakernia, O. (1998). Evolutionary design of a helicopter autopilot, In: *Advances in Soft Computing - Engineering Design and Manufacturing*, pp. 201-214, Springer-Verlag.
- Jakobi, N.; Husbands, P. & Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics, *Proceedings of the European Conference on Artificial Life*, pp. 704-720, Granada, Spain, June 1995.
- Keymeulen, D.; Iwata, M.; Konaka, K.; Suzuki, R.; Kuniyoshi, Y. & Higuchi, T. (1998). Off-life model-free and on-line model-based evolution for tracking navigation using evolvable hardware, *Proceedings of the European Workshop on Evolutionary Robotics*, pp. 211-226, Paris, France, April 1998.
- Koza, J. (1992). *Genetic Programming*. MIT Press, Cambridge, Massachusetts.
- Lee, W.-P. & Hallam, J. (1999). Evolving reliable and robust controllers for real robots by genetic programming. *Soft Computing*, Vol. 3, No. 2, 63-75.
- Lund, H.H. & Hallam, J. (1997). Evolving sufficient robot controllers, *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp. 495-499, Indianapolis, Indiana, April 1997.
- Marin, J.A.; Radtke, R.; Innis, D.; Barr, D.R. & Schultz, A.C. (1999). Using a genetic algorithm to develop rules to guide unmanned aerial vehicles, *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pp. 1055-1060, Tokyo, Japan, October 1999.
- Mattos, L.S. (2003). *The EvBot II*. Master's thesis, North Carolina State University, Raleigh, North Carolina.
- Meyer, J.-A.; Doncieux, S.; Filliat, D. & Guillot, A. (2003). Evolutionary approaches to neural control of rolling, walking, swimming and flying animats or robots, In: *Biologically Inspired Robot Behavior Engineering*, pp. 1-43, Physica-Verlag.
- Nelson, A.L. (2003). *Competitive relative performance and fitness selection for evolutionary robotics*. Ph.D. thesis, North Carolina State University, Raleigh, North Carolina.
- Nelson, A.L.; Grant, E.; Barlow, G. & White, M. (2003). Evolution of complex autonomous robot behaviors using competitive fitness, *Proceedings of the IEEE International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, pp. 145-150, Boston, Massachusetts, October 2003.
- Nolfi, S. & Floreano, D. (2000). *Evolutionary robotics*. MIT Press, Cambridge, Massachusetts.
- Nolfi, S.; Floreano, D.; Miglino, O. & Mondada, F. (1994). How to evolve autonomous robots: Different approaches in evolutionary robotics, *Proceedings of the International Workshop on the Synthesis and Simulation of Living Systems*, pp. 190-197, Boston, Massachusetts, July 1994.
- Oh, C.K. & Barlow, G.J. (2004). Autonomous controller design for unmanned aerial vehicles using multi-objective genetic programming, *Proceedings of the Congress on Evolutionary Computation*, pp. 1538-1545, Portland, Oregon, June 2004.
- Oh, C.K.; Cowart, G. & Ridder, J. (2004). Autonomous navigation control of UAVs using genetic programming, In: *NRL Review 2004*, pp. 197-199.

- Richards, M.D.; Whitley, D.; Beveridge, J.R.; Mytkowicz, T.; Nguyen, D. & Rome, D. (2005). Evolving cooperative strategies for UAV teams, *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1721-1728, Washington, DC, June 2005.
- Shim, H.; Koo, T.J.; Hoffmann, F. & Sastry, S. (1998). A comprehensive study of control design for an autonomous helicopter, *Proceedings of the IEEE Conference on Decision and Control*, pp. 3653-3658, Tampa, Florida, December 1998.
- Walker, J.; Garrett, S. & Wilson, M. (2003). Evolving controllers for real robots: A survey of the literature. *Adaptive Behavior*, Vol. 11, No. 3, 179-203.
- Winkeler, J.F. & Manjunath, B.S. (1998). Incremental evolution in genetic programming, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 403-411, Madison, Wisconsin, July 1998.
- Wu, A.S.; Schultz, A.C. & Agah, A. (1999). Evolving control for distributed micro air vehicles, *Proceedings of the IEEE Conference on Computational Intelligence in Robotics and Automation*, pp. 174-179, Monterey, California, December 1999.

Application of Artificial Evolution to Obstacle Detection and Mobile Robot Control

Olivier Pauplin and Arnaud de La Fortelle

*National Institute for Research in Computer Science and Control (INRIA), IMARA Team
France*

1. Introduction

The Fly Algorithm is an evolutionary algorithm used for stereoscopic reconstruction. In the classical approach, a pair of stereo images is processed in order to extract 3-D information and to infer a representation of the scene. Conversely, the Fly Algorithm builds potential 3-D models of the scene and tests their consistency with the two stereo images. This chapter will present some recent improvements of the algorithm, and its concrete application to obstacle detection and avoidance in mobile robotics.

Section 2 presents the notion of individual approach in evolutionary algorithms and the principle of the Fly Algorithm. New genetic operators are introduced. Several internal parameters can drastically change the algorithm behaviour: this issue is the topic of section 3, where a Pareto multi-objective optimisation leads to the obtention of an efficient set of parameters.

Section 4 describes a real time application to automatic driving on an electrical vehicle of the IMARA Team (INRIA). We focus on stop/go control and on direction control in order to avoid obstacles in front of the vehicle. The methods used are explained and results are shown.

Finally, section 5 concludes the chapter and gives some ideas of future work to further improve the algorithm.

2. The Fly Algorithm

2.1 Presentation

Evolutionary algorithms (Holland, 1975; Goldberg, 1989; Rechenberg, 1994), inspired by the evolution of species by natural selection, reproduction and mutation, are now common and widely used optimisation methods. The usual goal of these optimisation methods is to search the extremum of a function – called fitness function, or objective function – that is to say the best individual of the population after a certain number of iterations. The so called individual approach (or Parisian approach) (Collet et al., 1999) considers the solution to a problem to be given not only by the best individual, but by the whole population, or a significant part of the population. That is made possible by an appropriate formulation of the problem, splitting the representation of the object to be optimised into smaller primitives evolved separately.

The Fly Algorithm (Louchet, 2000; Boumaza & Louchet, 2003; Pauplin et al., 2005) is an evolutionary algorithm based on the individual approach, and used in the domain of computer vision (Jähne, 1999). The aim of the algorithm is to drive the population of individuals, defined as 3-D points (the “flies”) in front of a pair of cameras, into suitable areas of the search space, corresponding to the surfaces of objects present in the scene. The search space where the flies evolve is the intersection of the two cameras’ field of view, as shown on figure 1.

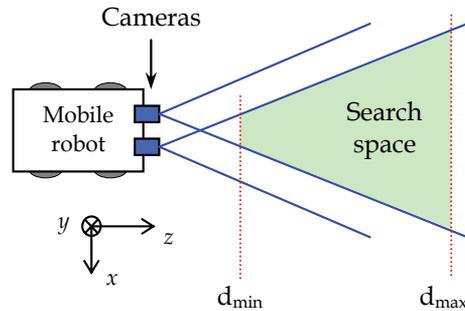


Figure 1. Example of device using the Fly Algorithm (top view)

The population of flies is initialised at random in the search space, and then evolves following the steps of an evolutionary algorithm.

2.2 Evaluation

The fitness function used to evaluate a fly compares its projections on the left and right images given by the cameras. If the fly is on an object’s surface, the projections will have similar neighbourhoods on both images and hence this fly will be attributed a high fitness.

The mathematical expression of the fitness function is:

$$F(fly) = \frac{|\nabla_x(M_L)| \cdot |\nabla_x(M_R)|}{\sum_{colours} \sum_{(i,j) \in N} [L(x_L+i, y_L+j) - R(x_R+i, y_R+j)]^2} \quad (1)$$

where:

- (x_L, y_L) and (x_R, y_R) are the coordinates of the projections of the current individual in left and right images
- $L(x_L+i, y_L+j)$ is the grey value at the left image at pixel (x_L+i, y_L+j) , similarly with R for the right image
- N is a neighbourhood around the projection of each fly, introduced to obtain a more discriminating comparison of the flies
- $|\nabla_x(M_L)|$ and $|\nabla_x(M_R)|$ are absolute values of the horizontal component of Sobel gradient on M_L and M_R , which are the projections of the fly in left and right images. That is intended to penalise flies which project onto uniform regions, i.e. less significant flies.

More details about the fitness function used can be found in (Pauplin et al., 2005; Pauplin, 2007).

2.3 Sharing and selection

If nothing is done to prevent it, flies tend to gather around a unique point of the search space. Such a population does not give a 3-D description of the scene. In order to force flies to explore the whole search space, a sharing (Boumaza, 2001) reduces the fitness of flies packed together.

The sharing is applied to the projections of the flies in the left image. A grid divides the left image into squares (figure 2), and the new fitness is given by:

$$F_{sharing}(fly) = F(fly) - n \times C_{sharing} \quad (2)$$

where n is the number of flies projecting into the same square as the considered fly, and $C_{sharing}$ (sharing coefficient) is a coefficient whose value can be tuned experimentally. A high sharing coefficient prevents areas in the search space to be unexplored, but also results in a lower average fitness.

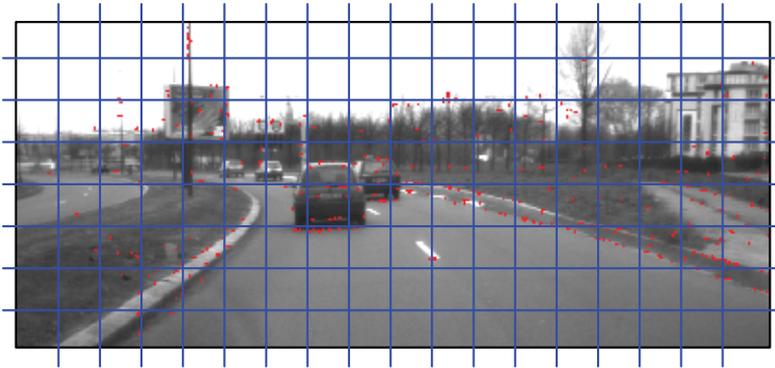


Figure 2. The grid used to apply the sharing. Flies appear as red dots

The selection is based on the values of $F_{sharing}$. The population is ranked according to $F_{sharing}$ and the best half is kept and forms the population of parents ($S/2$ individuals, S being the size of the population). $S/2$ children must be created.

2.4 Genetic operators

The population of offspring is created by applying one of five genetic operators $S/2$ times - each operator creates one child at a time. For each child, which genetic operator will be used is determined according to probabilities ($p_i, i = 1 \dots 5$) assigned to each operator.

The five genetic operators are described hereafter.

- Initialisation or "immigration". An individual is created at random. That ensures a constant exploration of the search space.
- Mutation. A new individual is created by adding random perturbations to a parent's coordinates. The random perturbations belong to the interval $[-i_m, i_m]$.
- Rank-scaled mutation. An individual is picked at random among the parents. The interval of mutation is modified as follows, in function of the rank r of the parent:

$$i_m \rightarrow i_m' = i_m \times \left(\frac{2 \cdot r}{S} + 0.5 \right) \quad (3)$$

The mutation operator is then applied with i_m' instead of i_m . That multiplies by two the interval of mutation for the parent with the lowest fitness, and divides it by two for the best parent. It can be indeed interesting to create new individuals close to the better parents. Conversely, a parent with a low rank is probably not on an object's surface, and it is probably not efficient to create an individual in this area.

- Elitist mutation. Two new individuals are created by mutation. Only the best of the two children is kept. Unlike the usual mutation, this elitist mutation is not a simple exploration of the search space but effectively tends to increase the average fitness of the population. However, it requires to evaluate two individuals, which may be a drawback for a real-time application.
- Crossover. The crossover can be useful to detect flat surfaces or oblong objects. Three individuals are picked at random in the best half of the parents, and the two closest of these three individuals are combined to produce one child. That aims to rarefy crossovers between far away parents, which are probably not on the same object, or between parents which are not on an object at all. The child is obtained as a classical linear combination of the coordinates of the two parents:

$$fly_1 \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \times fly_2 \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} \rightarrow fly_3 \begin{pmatrix} \lambda \cdot x_1 + (1-\lambda) \cdot x_2 \\ \lambda \cdot y_1 + (1-\lambda) \cdot y_2 \\ \lambda \cdot z_1 + (1-\lambda) \cdot z_2 \end{pmatrix} \quad (4)$$

where λ is a random number belonging to $[-0.1, 1.1]$. This interval is a bit larger than usual $[0, 1]$ weighting so as to avoid contraction: the potential underlying object can be larger than the segment linking the two parents.

3. Parameters adjustment

As in most evolutionary algorithms, the convergence and robustness of the algorithm depend on numerous internal parameters. Inappropriate values of parameters may for instance lead to a very slow convergence or to a premature convergence to a local optimum. In this section we will try to find a set of parameters allowing the fastest and most efficient convergence of the algorithm, in order to enable its use in real time obstacle detection and robot control.

The Fly Algorithm pursues two antagonist objectives:

- to increase the average fitness of the population
- to explore the whole search space.

Genetic operators tend to realise the first objective, whereas the sharing tends to realise the second one. However, the number of parameters makes it uncertain, if not impossible, to tune the parameters manually. A solution consists in batch-testing a great amount of sets of parameters and evaluating the results according to given objectives.

3.1 Objectives and parameters

The two objectives above-mentioned can be computed as follows.

- Average fitness of the best third of the population.
- Diversity of the population: we chose to measure the diversity using a grid similar to (but not identical to) the one used in the sharing, on the left image (cf. figure 2). The

number of horizontal divisions of the diversity grid has been set to 30, which enables an accuracy high enough for our applications (the view angle of our cameras is 42.5 degrees, hence the average angle corresponding to one division of the grid is 1.4 degrees). The diversity can be estimated as:

$$div(P) = \frac{\sum \min(Nf_{square}, Nf_{average})}{N_{squares}} \quad (5)$$

where:

- P is the population
- Nf_{square} is the number of flies in the considered square
- $N_{squares}$ is the total number of squares
- $Nf_{average} = S / N_{squares}$, S being the population size.

We chose eight parameters:

- Sharing coefficient $C_{sharing}$, varying in $[0, 0.65]$
- Number of horizontal divisions of the sharing grid, varying in $[5, 75]$
- Interval of mutation, varying in $[0, 2]$ (metres)
- Probabilities of the five genetic operators, p_i , $i = 1 \dots 5$, varying in $[0, 1]$ with $\sum p_i = 1$.

3.2 Pareto front

Generally, the optima of the different objectives are not reached for the same set of parameters. The aim of Pareto's method is to display a collection of optimal compromises between the objectives. These optimal solutions form the Pareto front.

In the case of a maximisation problem with l objectives, an individual X^* belongs to the Pareto front if there is no other solution X which verifies the condition:

$$\forall k \in [1, l], f_k(X) > f_k(X^*) \quad (6)$$

In other words, no other solution is better than X^* for all the objectives.

3.3 Procedure and results

200000 sets of parameters have been picked at random. The algorithm runs 0.25 seconds for one set of parameters, and then computes the two objectives corresponding to the final population; that is repeated nine times, so that the algorithm runs ten times for each set of parameters. The average of the ten values of fitness and diversity obtained gives the values of the two objectives for that set of parameters.

It is important to run the algorithm for a given duration, not a given number of iterations, as the length of iterations depends on the parameters. The size of the population is fixed to a standard value: $S = 3000$.

The results appear on figure 3: the performances of the 200000 sets of parameters are represented in the space of objectives. The Pareto front (green) is made of 237 sets of parameters, which are the best compromises found for these two objectives.

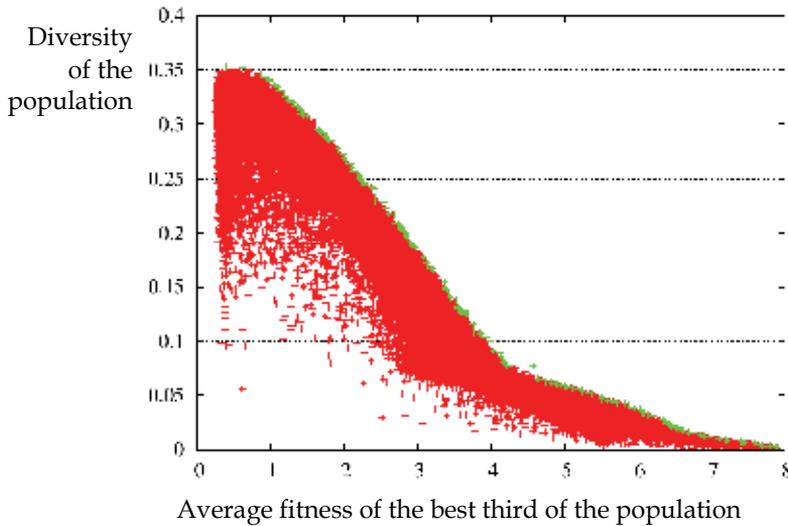


Figure 3. 200000 sets of parameters in the space of objectives. 237 sets of parameters form the Pareto front

It is difficult to justify the choice of one set of parameters belonging to the Pareto front instead of another. In the context of obstacle detection and automatic driving, diversity is mainly a way not to miss a new object entering the search space. Moreover, the density of flies must be significantly different in the areas where an object is present and in the empty areas. Hence, after several tests, it appeared that sets of parameters located on the “high fitness” side of the Pareto front were far better adapted to our applications than sets of parameters on the “high diversity” side of the Pareto front. We have finally chosen the compromise presented in table 1, which is located on the Pareto front with an average fitness around 5.

These parameters have proved experimentally to be much more efficient than other parameters manually tuned. They have been used to realise the experiments presented in section 4.

Sharing coefficient:	0.15
Number of horizontal divisions of the sharing grid:	30
Interval of mutation:	0.35 m
Probabilities to apply genetic operators:	
Initialisation	0.15
Crossover	0.27
Elitist mutation	0.23
Mutation	0.15
Rank-scaled mutation	0.2

Table 1. Chosen set of parameters

4. Application to mobile robotics

4.1 Material

The IMARA team (INRIA, France) owns electrical vehicles – CyCabs – which can be driven manually or automatically. A pair of cameras has been installed in front of a CyCab and connected to an embedded PC on which runs the Fly Algorithm. That PC is connected to CyCab controllers and can send them speed or direction instructions.

The two cameras provide rectified stereo images so that the three axes of each camera are parallel two by two, and their optical centers belong to the same horizontal axis (x). As a result, the computation time to calculate the projections of flies in left and right images is minimal.

The computer used is a standard PC under Windows XP, 2 GHz, 1 GB RAM.



Figure 4. The CyCab

4.2 Collision avoidance

The aim is to deliver a stop order when an obstacle appears close enough in the field of vision, in order to avoid frontal collision. The general idea is to see each fly as the source of a “warning value”, higher when:

- the fly is in front of the vehicle ($|x|$ small)
- the fly is near the vehicle (z small)
- the fly has a high fitness value.

The average of the warning values of the best part of the population gives a global indication of the probability that an obstacle is in front of the vehicle and that a collision could happen. A threshold can be experimentally determined, beyond which a stop signal has to be sent.

The function used to assign a warning value to each fly is made of three factors corresponding to the three conditions above:

$$w(\text{fly}) = w_{\text{space}}(x, z) \times F^\gamma \quad (7)$$

$$w_{space}(x, z) = \frac{10}{10 + x^4} \times \frac{100}{100 + \left(\frac{z}{\beta}\right)^4} \quad (8)$$

where x , z and F are the coordinates and the fitness value of the fly.

$w_{space}(x, z)$ gives the weight of the fly in the global warning value according to its position in the 3-D space. It does not have to depend on y as the search space has been vertically limited between the ground and the height of the CyCab. The parameter β allows to set the distance (on z) at which an object is considered as a potential risk (figure 5).

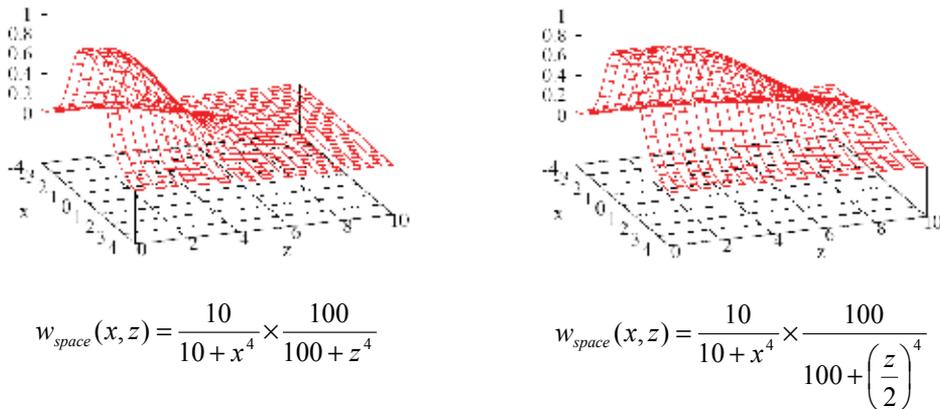


Figure 5. Influence of the parameter β (left: $\beta = 1$, right: $\beta = 2$)

Given the relatively small speed of a CyCab, around 2 m/s, we wish to distinguish between obstacles closer than $z = 5$ m (for $x = 0$) and those beyond $z = 5$ m. That is achieved when the value of β is such that the function

$$w_{space}(z)_{x=0} = \frac{100}{100 + \left(\frac{z}{\beta}\right)^4} \quad (9)$$

has a slope which is maximal (in absolute value) when $z = 5$ m. The maximal slope in absolute value of $w_{space}(z)_{x=0}$ is in $z = z_0$ given by

$$\beta = \frac{z_0}{\sqrt{10}} \quad (10)$$

For our application, $z_0 = 5$ m so we can take $\beta \approx 1.58$.

Another parameter from equation 7 is γ , the exponent of the fitness value. A small value for γ would favour flies with a small fitness, which are not very reliable. On the other hand, a high value for γ would favour better flies but could make the global warning value

depend on a few individuals with a high fitness, which would considerably increase the noise of the warning, as a small variation in a fitness value would result in a large variation in the warning value.



Figure 6. Scenes used to determine γ

The value of γ has been determined experimentally, considering the ratio between the average warning value when an obstacle is at 4 m and the average warning value when an obstacle is at 6 m, on the one hand, and the ratio between these values when an obstacle is at 4 m and when no immediate obstacle is present. Figure 7 and 8 show these ratios in function of γ . The scenes used for the experiment are shown on figure 6.

The curves on figure 7 and 8 have a different aspect, because when a pedestrian is present at 4 or 6 m the flies gather on him (with high fitness values), whereas when no obstacle is present the flies have a roughly uniform density in the space (with very low fitness values). It is then possible that, when $\gamma = 0$, the average value of $w_{space}(x,z)$ in the population is higher in the case of an empty scene than when an obstacle is present at 6 m. That shows that figure 7 is not enough to determine γ .

The value of γ is a compromise fulfilling the following conditions:

- The average warning value of a scene showing no obstacle must be lower than the one of a scene showing an obstacle at 6 m.
- The average warning value of a scene showing an obstacle at 6 m must be lower than the one of a scene showing an obstacle at 4 m.
- The warning values obtained on the scenes of figure 6 must be significantly different, in particular the “min” value on figure 7 should not be too close to 1.

According to figure 7 and figure 8, γ can be chosen between 1 and 2. That has been confirmed experimentally on the CyCab. We also checked that the value $\gamma = 4$ makes the CyCab stop inopportunately.

Results of individual warning values are shown on figure 9. Figure 10 shows the evolution of the global warning value when the CyCab moves towards a pedestrian; during the first 12.3 seconds, no obstacle is detected (the pedestrian is too far), and the global warning value is close to 0. The decision to stop the CyCab is given by a comparison of the global warning value with a threshold, for example 0.4 or 0.5.

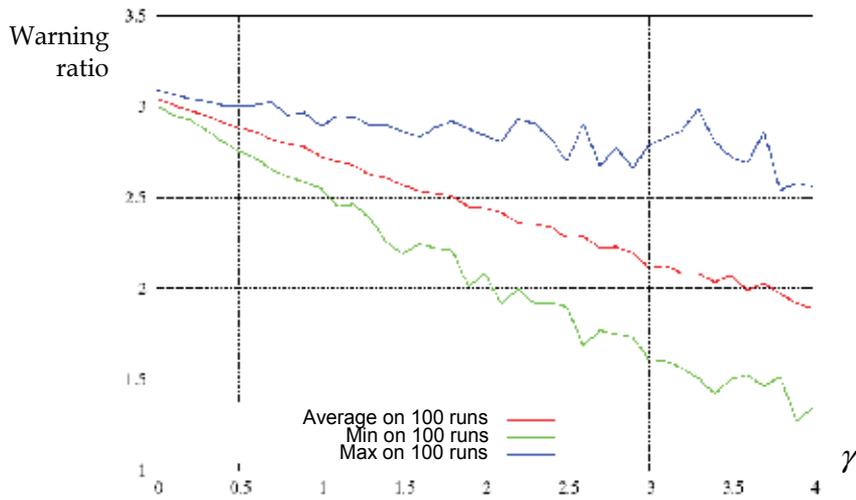


Figure 7. Average warning value obtained with an obstacle at 4 m divided by the average warning value obtained with an obstacle at 6 m

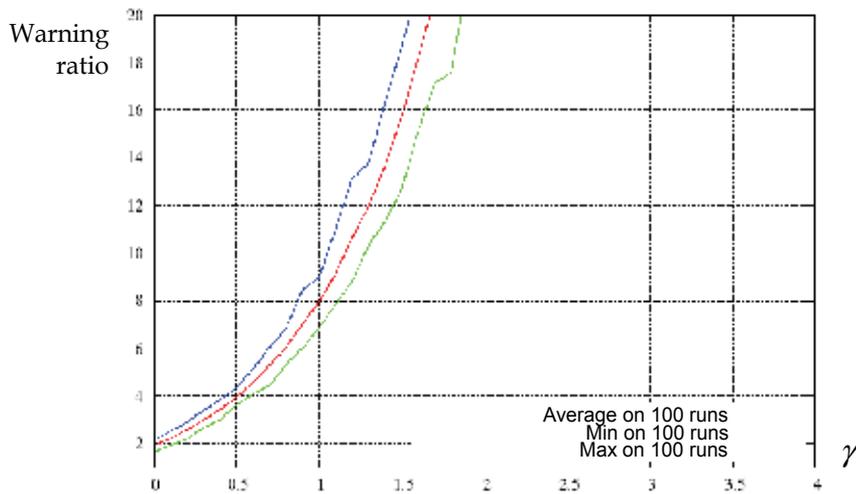


Figure 8. Average warning value obtained with an obstacle at 4 m divided by the average warning value obtained without any obstacle



Figure 9. Warning values in different situations. Left column: flies are represented as dots. Right column: flies appear as spots as dark as their warning value is high

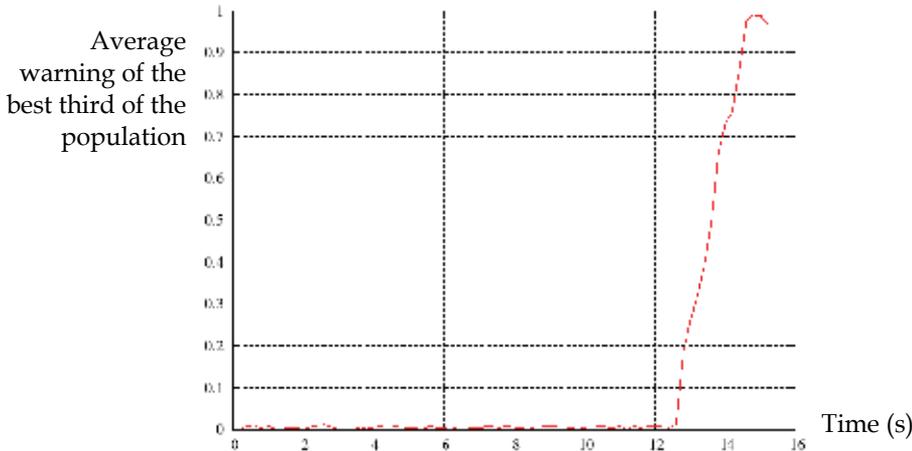


Figure 10. Evolution of the global warning value when the CyCab moves towards a pedestrian

4.3 Direction control

In some cases, objects are present in the scene but there is no immediate risk of collision. We can consider modifying the trajectory of the robot so that it will move forward staying as far as possible of the objects. The CyCab being a non holonom robot, it can not change its orientation without moving forward (or backward). The first signal to be sent to the CyCab controllers, after processing the actual population of flies, is the stop/go signal based on the global warning value (cf. previous section). If that signal is a “go”, a direction control is computed and sent to the wheels’ controllers.

The method we use is based on the warning values. The projection on the horizontal plan of the search space is divided into $N_{sectors}$ sectors centered on the intersection of the two cameras’ field of view, as described by figure 11.

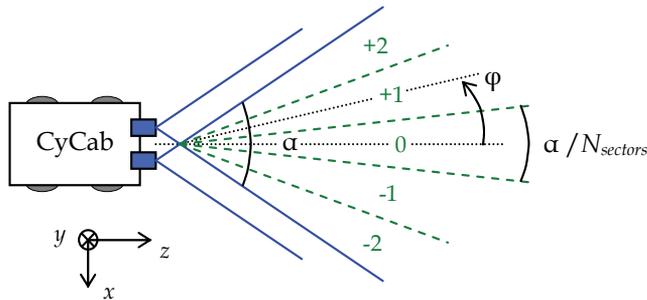


Figure 11. Example of division of the space in five sectors ($N_{sectors} = 5$)

The average warning value is computed for each sector, which gives a measure of the obstruction in each corresponding direction. Those values are compared to a threshold, and those under the threshold are considered equal to zero. This threshold, relatively low, is necessary for the case where no obstacle would be present. The number $N_{sectors}$ is a compromise: too small a number of sectors makes the representation of the obstruction imprecise, and too high a number makes it less reliable (the number of flies in each sector

must be high enough to give a reliable information). We have obtained good experimental results with five or seven sectors. It is appropriate to take an odd number of sectors, so it is possible to go straight ahead.

Let φ be the angle of the direction where to go, $\varphi = 0$ corresponding to the "straight" direction, as shown on figure 11. We tested two strategies.

- Strategy 1: the CyCab goes in the direction corresponding to the sector with the smallest warning value. If several sectors have a warning value equal to zero (which happens frequently due to the threshold), the direction chosen among these sectors is the closest to $\varphi = 0$.
- Strategy 2: the CyCab goes in the direction "opposed" (see below) to the sector corresponding to the highest warning value. If the warning values of each sector are null, the CyCab goes straight. If the highest warning value is in the direction $\varphi = 0$, the CyCab goes on the very left or the very right depending on the warning values in these directions.

Let Φ be the direction of the highest warning value. The opposed direction is defined by:

$$\varphi_{opposed} = \Phi + \alpha \cdot \frac{N_{sectors} + 1}{2 \cdot N_{sectors}} \quad \text{if } \Phi < 0 \quad (11)$$

$$\varphi_{opposed} = \Phi - \alpha \cdot \frac{N_{sectors} + 1}{2 \cdot N_{sectors}} \quad \text{if } \Phi > 0$$

α being the horizontal view angle. For instance, on figure 11, the direction opposed to "+2" is the direction "-1", the direction opposed to "+1" is the direction "-2" (and vice versa).

Both strategies have been tested on a CyCab. The strategy 2 has proved more satisfying. The strategy 1 gets easily trapped, specially if $N_{sectors}$ is high (7 or more). For example, if the highest warning value is in direction "+1", the CyCab will go straight until it is stopped by the anti-collision signal. The strategy 2 is more active (the CyCab goes straight only if no obstacle is detected), which results in a greater efficiency for obstacle avoidance.

Three steps of the trajectory of a CyCab controlled by strategy 2 are shown on figure 12.

- Step (a): a pedestrian is detected on the right. The CyCab starts to turn left.
- Step (b): the border of the road is now detected on the left. The CyCab starts to turn right.
- Step (c): no more obstacle. The CyCab goes straight.

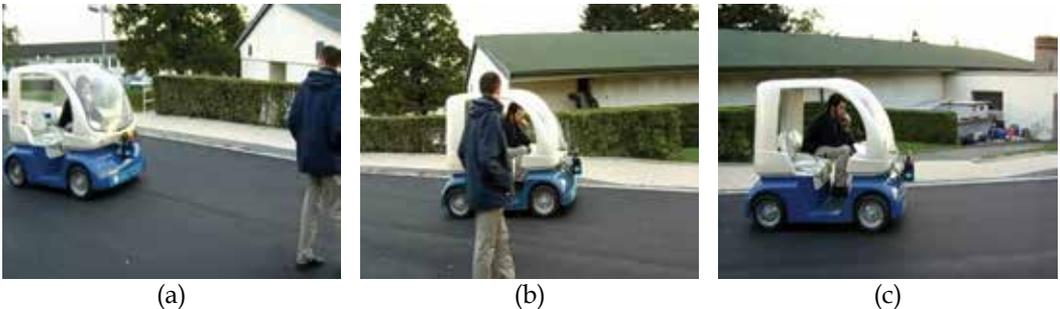


Figure 12. CyCab controlled by strategy 2

5. Conclusion

The Fly Algorithm embedded in a CyCab is able to detect obstacles, and to compute stop/go and direction controls accordingly, in real time. That is largely due to the optimisation of the efficiency conducted in section 3. It is also due to the fact that we have voluntarily stayed close to the natural output of the algorithm – a cloud of 3-D points – and have used it directly, without any prior processing. The control strategies tested are very simple and may be improved.

Future work includes speeding up the frame processing using CMOS sensors – which may be well adapted to the computation of the fitness of the flies - instead of CCD, and to increase the speed using FPGA in the evaluation part of the evolutionary algorithm. Concerning the algorithmic part, we could consider adapting dynamically the search space according to the application or the conditions (e.g. the speed of the robot). Other ways to enhance the algorithm could be to change the set of parameters during the convergence period (a bit like Simulated Annealing), and to change the paradigm (at the moment: use a lot of very simple features, here 3D-points) and to use more complex features with dynamics adapted to the use. This is then closer to swarm work. But it could also offer a better interaction with more classical obstacle detection/classification: use the Fly Algorithm to detect region of interest within which dedicated algorithm would refine the detection. An open problem is then: can we also use this detection to enhance the Fly Algorithm runs?

6. References

- Boumaza, A. & Louchet, J. (2001). Dynamic Flies: Using Real-Time Parisian Evolution in Robotics, *Proceedings of EVOIASP '01*, Lake Como, Italy, April 2001
- Boumaza, A. & Louchet, J. (2003). Robot Perception Using Flies. *Proceedings of SETIT'03*, Sousse, Tunisia, March 2003
- Collet, P.; Lutton, E.; Raynal, F. & Schoenauer, M. (1999). Individual GP: an Alternative Viewpoint for the Resolution of Complex Problems. *Proceedings of GECCO'99*, Orlando, USA, July 1999, Morgan Kaufmann, San Francisco
- Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 0201157675, Boston, MA, USA
- Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 0262581116, Ann Arbor, MI, USA
- Jähne, B.; Haussecker, H. & Geissler, P. (1999). *Handbook of Computer Vision and Applications*. Academic Press, 0123797705, San Diego, CA, USA
- Louchet, J. (2000). Stereo Analysis Using Individual Evolution Strategy. *Proceedings of ICPR'00*, Barcelona, Spain, September 2000
- Pauplin, O.; Louchet, J.; Lutton, E. & de La Fortelle, A. (2005). Evolutionary Optimisation for Obstacle Detection and Avoidance in Mobile Robotics. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 9, 6, (November 2005), pp 622-629
- Pauplin, O. (2007). Application de techniques d'évolution artificielle à la stéréovision en robotique mobile autonome. PhD thesis, Université Paris Descartes, Paris, 2007
- Rechenberg, I. (1994). *Computational Intelligence: Imitating Life*. IEEE Press, 0780311043, Piscataway, NJ, USA

Hunting in an Environment Containing Obstacles: A Combinatory Study of Incremental Evolution and Co-evolutionary Approaches

Ioannis Mermigkis and Loukas Petrou

Aristotle University of Thessaloniki, Faculty of Engineering, Department of Electrical and Computer Engineering, Division of Electronic and Computer Engineering, Thessaloniki Greece

1. Introduction

The field of evolutionary robotics has drawn much attention over the last decade. Using a very general methodology (Evolutionary Computation –EC) and with minimal supervision, it is possible to create robotic controllers that cover a vast repertoire of behaviors, either in simulation or real environments, for commercial, pure research or even entertainment purposes.

The strong point of evolutionary robotics is that if the fitness criterion is defined properly, it is possible to evolve the desired behavior regardless (or at least in a big degree) of other parameters such as Genetic algorithms properties (population size, mutation type, selection function) or even controller specific properties (in case of neural networks, even the architecture can prove irrelevant to the success of the algorithm).

An important feature is the ability of Evolutionary Algorithms (EAs) to find solution simpler than the corresponding hand-made ones. For example, in a garbage collection task, Nolfi (1997) discovered that the Genetic Algorithm (GA) evolved the network to use two distinct modules for a task that hand-crafted controllers would need to define four.

This ability however shows also the limitations of EAs to tasks that are simple in concept. If the problem requires a set of behaviors to be available and switch between one another, a simple GA will not find a successful solution. For this reason, a collection of techniques named Incremental Evolution have been developed to create the possibility of evolving multiple behaviors in one evolutionary experiment.

We shall attempt to evolve behaviors on two competing species in predator-prey setup for simulated Khepera (K-team, 1995) robots, in an area containing obstacles. The robotic controllers will be discrete time recurrent neural networks of fixed architecture and the synaptic weights will be subject to evolution. The evolutionary algorithm will be a standard GA with real value encoding of the neural synapses and mutation probability of 5% per synapse. The experiments will run exclusively on simulation, the Yet Another Khepera Simulator (Carlsson & Ziemke, 2001) respectively. The experimental setup, network architectures and genetic algorithms will be presented in detail in the following sections.

The chapter's structure is the following: Incremental evolution is defined in section 2 and the basic guidelines for successful fitness function definition are enumerated in evolutionary and co-evolutionary experiments. In section 3 the problem of hunting (and evading predator) in an environment that requires to avoid obstacles as well is presented. This problem requires combination of techniques as it requires various behavioral elements. Section 4 describes the setup of the experiments, regarding environmental elements, robotic sensors and actuators, robotic neural controllers and the genetic algorithm. It is also analyzed what challenges poses this environment compared to an empty arena and the cases of contradicting readings of the various sensors (the perpetual aliasing (Whitehead & Ballard, 1991) and poverty of stimulus problems).

Sections 5 and 6 present the results of the various experiments defined in section 4. In section 5 the behavioral elements that can be observed by looking at five best individuals are described while in section 6 the data taken from fitness (instantaneous and master) evaluation are presented in order to validate hypotheses made.

Section 7 concludes the chapter and future work is proposed in section 8.

2. Incremental evolution

The first problem that faces every evolutionary procedure is the so-called bootstrap problem: in the first generations of evolution, where best individuals are mainly an outcome of random generation, it is quite unlikely that individuals shall evolve to get an adequate fitness score that can discriminate them from the rest of the population.

Incremental evolution can cope with this kind of problems by describing a process of gradually (incrementally) hardening the evolutionary problem. This can be achieved with various ways

- a) By directly linking the generation number to the fitness function, e.g. add more desired tasks to the overall fitness evaluation.
- b) By making the environment harder as generations pass, e.g. add more obstacles in a collision free navigation task.
- c) In the commonly used case of evolutionary neural training, train the network for one task and then use the final generation of the first pass as the initial generation to train the second task.

Nolfi and Floreano (2000) have stated that while incremental evolution can deal sufficiently with the bootstrap problem, it cannot operate in unsupervised mode and violates their principal of fitness definition being implicit. As the supervisor must define every step of the process, evolution cannot run unsupervised and scale up to better and unexpected solutions.

This argument confines the usability of incremental evolution to small, well-defined tasks. While this is a drawback for the theoretical evolutionary robotics, that visualize evolutionary runs that can go-on for millions of generations and produce complex supersets of behaviors, while being unattended, real robotic problems encourage the incorporation of small behavioral modules in larger, man-engineered schemas. These modules can be produced using several methods and evolutionary algorithms are as good as any. Togelius (2004) invented a method called incremental modular in which he defined modules in subsumption architecture (Brooks, 1999). The interconnection between modules was pre-defined and fitness evaluation proceeded for the whole system, while neural network evolved simultaneously.

2.1 Guidelines to design successful fitness functions

Designing the proper fitness function is fundamental for the success of the evolutionary procedure. While Genetic Algorithms (GAs) and other evolutionary methodologies work as optimization methods for the given fitness function, the definition of the proper function for the task at hand requires a lot of work. In previous article (Mermigkis & Petrou, 2006) we have investigated how the variation in the fitness function can produce different behaviors while the other parameters (network architecture and size, mutation rate, number of generations and epochs) remain the same.

In evolutionary systems, it has been stated that fitness functions should be as simple as possible (implicit) and describe the desired behavior rather than details about how to be achieved (behavioral). It is also better to calculate the fitness based only on data that the agent itself can gather (internal). This allows the evolutionary procedure to continue outside the pre-defined environment of the initial experiment and continue the evolution in real environments where external measurement isn't possible. These three qualities have been summarized by Nolfi and Floreano (2000) in the conception of fitness space.

2.2 Incremental evolution and coevolution

Several research groups have pointed out that evolving two species one against each other is a form of incremental evolution which falls into case (b) of the previous paragraph: If the competing species is considered part of the environment for the one species, then the progress of its fitness is considered hardening of the environment for the opponent and vice versa. This could work flawlessly if there hadn't been the phenomenon of cyclic rediscoveries, both reported in evolutionary robotics (Cliff & Miller, 1995a, 1995b, 1995c, Floreano and Nolfi, 1997) and evolutionary biology (Dawkins, 1996). Cyclic rediscovery, also known as red queen effect, is the tendency of competing species to develop qualities of previous generations in later stages, because these qualities can cope better with the opponent of the current generation. While several methodologies have been proposed to overcome this problem, such as hall of fame tournaments, the problem still exists in nowadays implementations.

3. Hunting in an environment that contains obstacles

The Predator - prey or hunt situation has been explored by different research groups (Mermigkis & Petrou, 2006), (Cliff & Miller, 1995a), (Floreano & Nolfi, 1997), (Buason & Ziemke, 2003), (Haynes & Sen, 1996) using different methodologies. However, in most cases the environment (or arena) of the experiment has been an empty space confined by walls with no obstacle contained within. In previous work (Mermigkis & Petrou, 2006) we explored the possibilities of such an experimental setup and watched the emergence of different kinds of behavior (and behavioral elements such as evasion, pursuit, lurking or pretence). In this paper we shall try to conduct the hunt experiment in an arena that contains square objects (Figure 1) and see how the emerging agents cope with this situation.

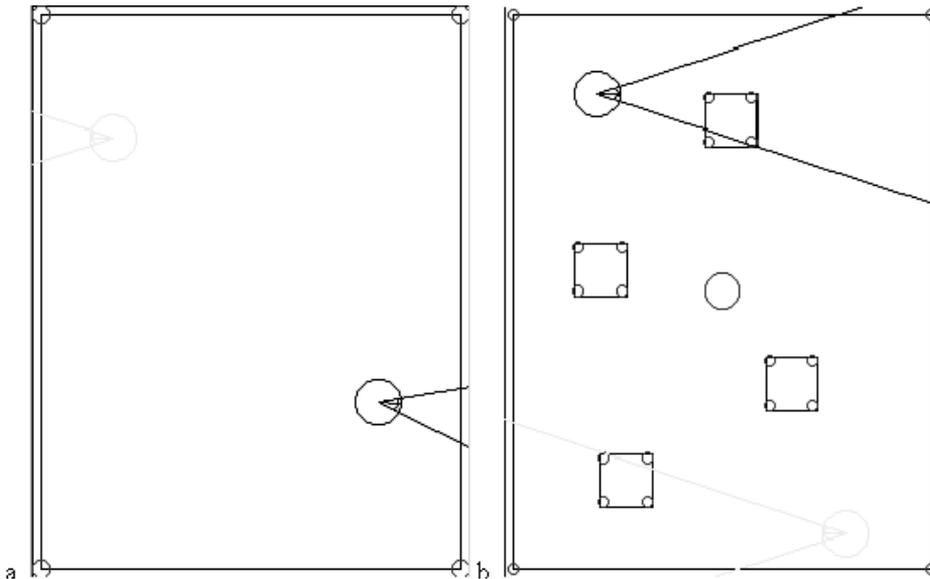


Figure 1. Arena and initial Positions. As every run consists of 4 epochs, the agents switch starting positions. A: Arena without obstacles. B: arena with obstacles

3.1 Need for simulation

The experiments concern the co-evolution of two simulated Khepera robotic vehicles. One vehicle (predator) evolves trying to catch the opponent (prey) while the prey's evolutionary target is to wander around the arena avoiding collisions with obstacles and the predator. YAKS (Yet another Khepera Simulator) (Carlson & Ziemke, 2001) has been adopted to simulate the robotic environment. The reason why simulation has been used is time restrictions: In the following chapter several experiments are conducted that last for 500 generations of 100 individuals. This leads to many hours of experiments that have to be spent and simulation is the best way to a) parallelize the conduction of experiments by spreading to several PCs and b) simulation is in general faster than conducting the experiment with real robots.

On the other hand, various research groups (Carlson & Ziemke, 2001), (Miglino et al., 1995), (Jacobi et al., 1995) have shown that it is possible to evolve behaviors in simulation that can easily be transferred to real robots in few more evolutionary runs.

4. Experimental setup

4.1 Calculating fitness

Experiments are conducted in the arena depicted in figure 1. Fitness is evaluated in 4 epochs of 200 simulated motor cycles. In every epoch the two agents switch starting positions in order to eliminate any possible advantage by the starting position.

The Evolutionary algorithm (EA) adopted is a simple Genetic Algorithm (GA) applied on Neural Networks (NN) of fixed architecture. Christensen and Dorigo (2006) have shown that other EAs such as the (μ, λ) Evolutionary Strategy can outperform the Simple GA in incremental tasks, however we try to follow the experimental framework of (Mermigkis &

Petrou, 2006) in order to be able to make comparisons. In the same spirit, only mutation is applied on individuals.

```

Main{
  Generation 0:
    Create random populations A,B
    Calculate fitness A against individual B[0]
    Sort pop A (fitness(A[0])=max)
    Calculate fitness B against A[0]
    Sort pop B
    Hall_of_Fame_A[0]=A[0]
    Hall_of_Fame_B[0]=B[0]
  Main GA Loop:
    for (generation=1;generation<nrOfGenerations;generations++){
      A'=create_new_gen(A)
      calculate fitness A' against B[0]
      Sort A'
      B'=create_new_gen(B)
      calculate fitness B' against A'[0]
      Sort B'
      Hall_of_Fame_A[generation]=A'[0]
      Hall_of_Fame_B[generation]=B'[0]
      A=A',B=B'
    }
  }
  create_new_generation(pop){
    for (elite=0;elite<nrOfElites;elite++){
      for (mut_ind=0;mut_ind<nrOfIndivids/nrOfElites;mut_ind++){
        pop[rOfElites+20*elite+mut_ind]=mutate(pop[elite])
      }
    }
  }
  mutate(individual){
    for (synapse=0;synapse<nrOfSynapses;synapse++){
      individual[synapse]+=mutation_probability*gauss_rand()
    }
  }
}
    
```

Listing 1. Pseudocode of the Genetic Algorithm

The experiments consist of two populations competing against each other for 500 generations. Each population consists of 100 individuals. Fitness of population A is calculated by competing against the best individual of population B of the previous generation or the 10 previous generations.

The Genetic algorithm followed is shown in Listing 1: First two random populations are created and are evaluated one vs. one. From every generation, the 5 best individuals are selected and passed to the next generation. The remaining 95 individuals are produced by mutated copies of the 5 selected ones (19 copies per elite individual). Real-value representation has been chosen since binary encoding constrains synaptic values to predefined min and max levels. Mutation is produced by adding to each synaptic value a random number from a Gaussian distribution multiplied by 0.05 (the mutation probability).

4.2 Agent Hardware and Neural Controllers

The simulated Kheperas originally used the 8 infrared sensors and a rod sensor. The rod sensor is a kind of camera of 10 pixel resolution that can locate other agents equipped with rods. It is assumed that the rods are high enough so that the rod sensor can detect a robot even if there is a wall or other obstacle in the middle.

In order to strike out accidental contacts between the vehicles we define that contact is made if the predator robot touches the prey with the central front part (prey must be in the 4th or 5th pixel of the predator's rod sensor).

The rod sensor however doesn't return any info about how far the other vehicle is, only the relative angle of the two. It is possible that if the two vehicles are in the opposite sides of an obstacle, then rod sensor indicates opponent's presence and infrared sensors indicate contact with something. While there have been studies (see (Nolfi & Floreano, 2000) chapter 5 for a comprehensive review) that show that simple NNs can differentiate between objects based only on IR sensory patterns, it is possible that the agent's controller cannot tell whether there has been contact with the opponent vehicle or an obstacle. For this reason we have conducted another series of experiments in which we have added light sources on top of the simulated vehicles. This way the vehicle can detect the proximity of an opponent by using the 8 ambient light sensors.

Also, since the desired behavior has two distinct elements (collision free movement and evasion-pursuit) we have experimented with a simple NN with a hidden layer of 4 recursively interconnected neurons and with recurrent connection on the output neurons, and a NN that contains a hidden layer of two modules (modular architecture). Each module consists of a decision neuron and 4 value neurons recurrently connected to each other. Hidden neuron values are propagated to the output neurons only if the decision neuron has a positive activation value. Figure 3 shows the architecture of the networks used in this experiment.

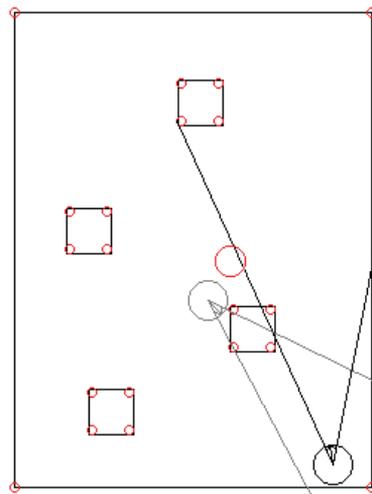


Figure 2. Predator robot (grey) stumbled into an obstacle considering it to be the prey (black)

The input layer of both networks contains one bias neuron that has fixed activation of 1.0 and neurons that map the several sensory inputs scaled so as the minimum value is 0 and the maximum 1.0. Hidden layer and output layer neuron activation function is the sigmoid

function while the decision neurons use the step function. Hence, the value y_j of output neuron j at time step t is given by equation (1)

$$y_j[t] = d_M[t] \cdot \left(\frac{1}{1 + e^{-A_j[t]}} - 0.5 \right) \tag{1}$$

$$d_M[t] = \begin{cases} 1, & A_M[t] > 0 \\ 0, & A_M[t] \leq 0 \end{cases} \tag{2}$$

$$A_j[t] = b_j + \sum_{i=0}^I w_{ij} x_i[t] + \sum_{k=0}^K w_{kj} y_k[t-1] \tag{3}$$

Where d_M is the activation value for the decision neuron of module M (if defined), x_i the value of input i , b_j the bias value for neuron j and w_{ij} the various weights for forward and recurrent connections.

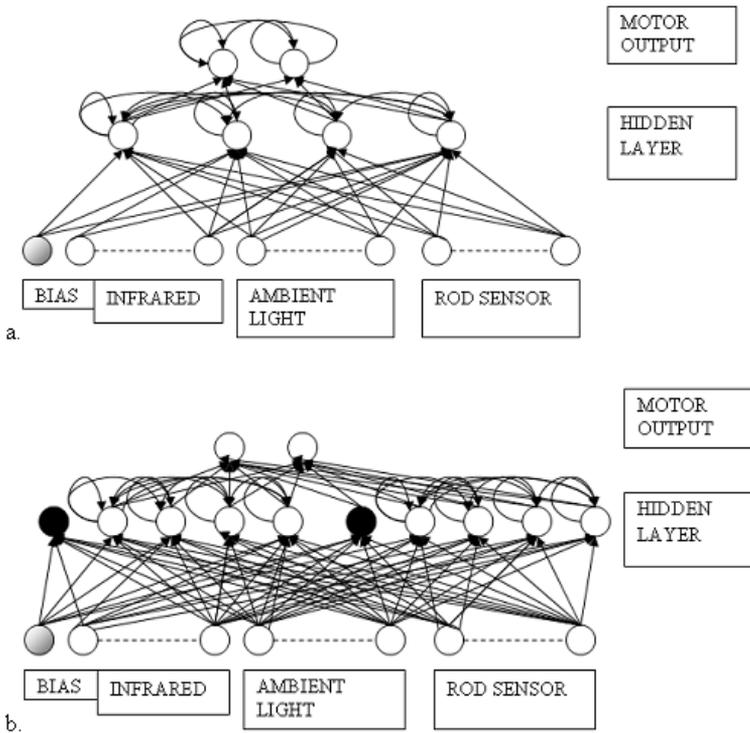


Figure 3. Network architectures tested a. Simple recurrent network with hidden layer of 4 neurons connected to each other. Ambient light input is not present in all experiments b. Hidden layer contains two modules with decision neuron. If decision neuron's activation is >0 then the module neuron's activation is propagated to the output. The output neurons are not recurrently connected

5. Evaluating co-evolution

5.1 Qualitative data in co-evolution

Two elements that are very common in co-evolutionary situation, both in evolutionary biology and evolutionary robotics are the arm-races and the cyclic rediscovery of strategies (a phenomenon commonly known as the red queen effect). The arm-races mean that as generations pass, opposing species constantly alter their strategies in order to beat their opponents. Arm-races can be depicted in instantaneous fitness graphs as oscillations which happen because a strategy x_1 that can beat an opposing strategy y_1 cannot beat strategy $y_2 > y_1$. Since evolutionary algorithms slightly change winning strategies, the x_2 strategy that competes against y_2 is more likely to loose.

Cliff and Miller (1995b) validated this phenomenon in robotic simulation experiments and concluded that the instantaneous fitness graphs are not adequate to show the progress of co-evolving populations. Instead, they proposed the CIAO (Current Individual vs. Ancestral Opponents) graphs. A CIAO graph is a grid of pixel where pixel (x,y) contains a color representation of the fitness score of species A generation x competing against Species B generation y .

In an ideal arms-race, an individual $x_2 > x_1$ that can beat an individual $y_2 > y_1$ should also be possible to beat y_1 as well, leading to CIAO graphs similar to Figure 4a and 4b. However both in nature and robotics this doesn't happen. It is possible that y_2 loses to x_1 . This means that it is likely that y_1 will re-appear as $y_3 > y_2$ in order to compete against x_1 that reappears as x_3 . This way y_2 will reappear again and the circle continues, leading to the phenomenon of cyclic rediscovery of strategies. CIAO graphs that correspond to the emergence of cyclic rediscovery have the tartan pattern similar to figure 4c.

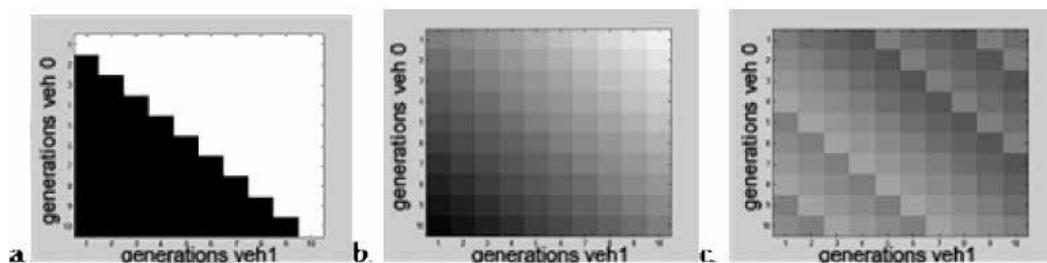


Figure 4 CIAO graphs patterns a: The idealized form for binary fitness function b: the idealized form for proportional fitness function c: the tartan patterns that indicate cyclic rediscovery of strategies

In order to reduce the Red Queen effect's impact, Nolfi and Floreano (1998) proposed the Hall of Fame tournament: Fitness of an individual x of Species A must not only be calculated against opponent of just the previous generation but also against more ancestral opponents. Ideally, fitness should be calculated against all ancestral opponents, in what Floreano and Nolfi call the Master tournament. However, such an evaluation can make the evolved task too hard and paralyze the evolutionary process, as no viable solution can be found. In the experiments presented here, fitness has been evaluated against previous generation best opponent (tournament depth 1) and against the champions of the previous 10 generations (tournament depth 10).

Since CIAO graphs can give only a rough icon of the evolutionary process, Floreano and Nolfi have proposed the Master Fitness measurement, which is the average fitness of an individual against opponents of all generations.

5.2 Methodological approaches of the Predator-Prey experiment

In order to evolve his virtual creatures, Sims (1994) conceptualized the idea of two populations competing against each-other. He experimented with various competition combinations (All vs. All, All vs. best of previous generation, random selection of opponent from previous generation) and concluded that the all vs. best schema could give the best results. Although he didn't analyze the dynamics of the evolutionary process per se, he reported that "*interesting inconsistencies*" could be reported, referring to similar agents evolving again. On the contrary, Cliff and Miller (1995a, 1995b, 1995c) studied co-evolution of predator and prey simulated robotic agents giving emphasis on game-theoretical and methodological aspects.

Floreano & Nolfi (1997), Nolfi and Floreano (1998), conducted similar experiments in real and simulated Khepera robotic miniatures and shown that the Red Queen Effect can be partially anticipated by calculating fitness against more ancestral opponent and not only the last one. This methodology provides more stable behaviors. They also experimented with arenas containing obstacles and drawn conclusions regarding the emergence of static behaviors like obstacle avoidance and how this affects the co-evolutionary process. Floreano et al. (2001) have also studied the possibility of neural networks that modify the synapses in runtime using Hebb rules, instead of the commonly used gene-coded synapses. They concluded that Hebbian modified neurons (or plastic neurons, as they call it) allow the predator to include a wider repertoire of behaviors.

Apart from the coevolutionary methodology (evolving population against the opponents best ancestors), the predator prey experiment has been handled using conventional evolutionary methodology. The most popular variation is that the predator behavior is evolved against prey of standard controller. E.g., Haynes and Sen (1996) used Genetic Programming (GP) to evolve a team of predators against a prey that had a hand-coded controller. Shultz et al (1996) conducted a variation of this experiment where the predator robots (here called the shepherds) tried to lead the prey (here sheep) into a predefined place in the arena (the corral). Potter et al. (2001) extended this experiment by adding another robotic agent (the fox). This experiment has rich dynamics as there are multiple objectives that must be optimized by the evolutionary process. Similar methodology has been used by Nietschke (2003) who evolved neural controllers for simulated Khepera vehicles if predator groups trying to immobilize a prey agent. The prey agent uses static obstacle avoidance controller while the predators could either share the same genotype, or evolve in parallel.

The paradigm of coevolving populations has been shown (Nolfi & Floreano, 1998) to give better solutions than the single evolution of predator behaviors. It also allows for game-theoretical assumptions to be made. However, the main disadvantage is the large number of fitness evaluations that need to take place using this methodology: Regarding the tournament's depth, the method needs $2 \times D \times G$ evolutionary runs, where G is the number of Generations and D the tournaments depth. Furthermore, if the experiment includes a third robotic species (i.e. the fox) or heterogeneous robotic teams, the fitness evaluation becomes too much complicated.

Other groups have issued the best combination of individuals to test against. Rosin and Belew (1996) have proposed the method of Shared Sampling in which not only the best individual but also individuals with rare genotypes are preserved in each generation.

Stanley and Miikkulainen (2004) and Gomez and Miikkulainen (1997) have conducted co-evolutionary experiments of Battle for food and Predator-Prey in an arena with obstacles respectively. They have used a methodology that evolves not only the network weights but the architecture as well.

6. Results and Findings

We have conducted experiments using various agent setups:

- A. recurrent neural network,
- B. recurrent network with ambient light sensory input,
- C. neural network that contains a hidden layer of two modules (modular architecture) with ambient light inputs.

For the three combinations we have tried the following experimental setups:

1. Fitness function (FF) of hunt experiment, defined by equations (4), where T is the total duration of the experiment and T_C the Time-to-Contact. Initial populations are randomly generated.

$$f_{PREY} = \frac{T_C}{T} f_{PREDATOR} = 1 - \frac{T_C}{T} \quad (4)$$

2. Prey's fitness function is a combination of collision-free navigation and evasion, as seen in equation (5), where v_1, v_2 are the motor velocities normalized in $[0,1]$ (0:full backward, 1:full forward), $\max(IR)$ the maximum value of IR sensors and T the total duration of the experiment and T_C the time until contact (with predator) is made. In simple Obstacle avoidance calculations $T_C=T$. Predator's FF is the same as in equation (4). Initial population is randomly generated.

$$f_{ov} = \frac{1}{T} \int_{t=0}^{T_C} I(v_1 + v_2 - 1) \sqrt{1 - |v_1 - v_2|} dt, I = \begin{cases} 1, \max(IR) < \frac{1000}{1023} \\ 0, \max(IR) \geq \frac{1000}{1023} \end{cases} \quad (5)$$

Equation (5) describes obstacle avoidance elements that favour forward movement, since the element (v_1+v_2-1) has value of -1 in case of full backward motion ($v_1=v_2=0$). We tried another variation where the element v_1+v_2-1 was replaced by its absolute value $|v_1+v_2-1|$ in order to include the possibility of backward motion but didn't observe the emergence of backwards motion (in prey robot). Hence, we didn't measure this case (of favouring backwards motion).

3. Using equations (4) , but with initial population being the last population of a previously run GA aiming to evolve collision free navigation with FF given in equation (5).

6.1 Behavioural elements – strategies and trajectories

From the methodological point of view, an easy mistake would be to present strategies categorized by the various fitness functions or vehicle controller. Since the task at hand and the environment is common, the successful solutions will be similar for all cases.

As can be seen by observing the produced behaviours, the problem to solve for both agents is complex and includes several distinct stages. A prey must scan the area avoiding been close to predator. The predator's problem has 2 distinct phases: First the predator must have

Predator's strategy also always included the rotate-on-spot tactic where the predator rotated until rod-sensor indicated prey in the visual field.

The obstacles of the arena pose new challenges to the evolutionary procedure. Moving in open circles is not a strategy that can consistently be followed as the obstacles will interrupt the open circle.

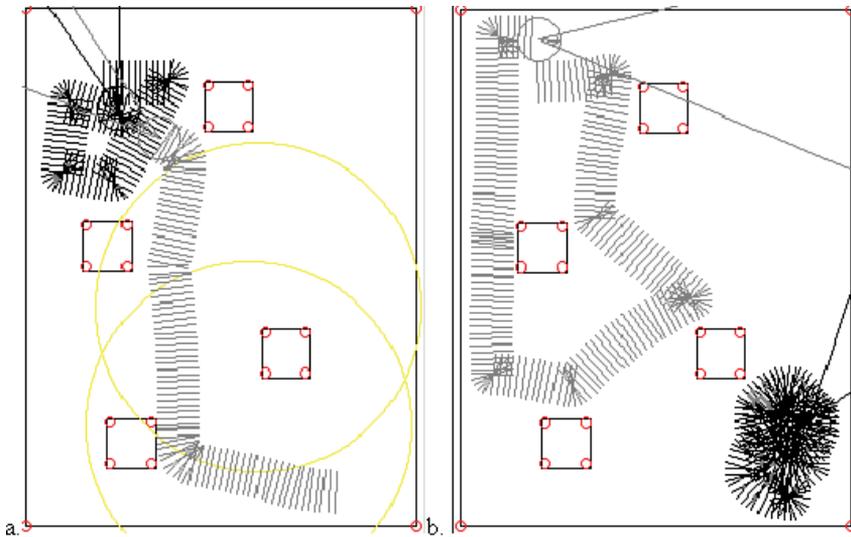


Figure 7. Using obstacles as landmarks: a. Successful approach, b. Unsuccessful guidance

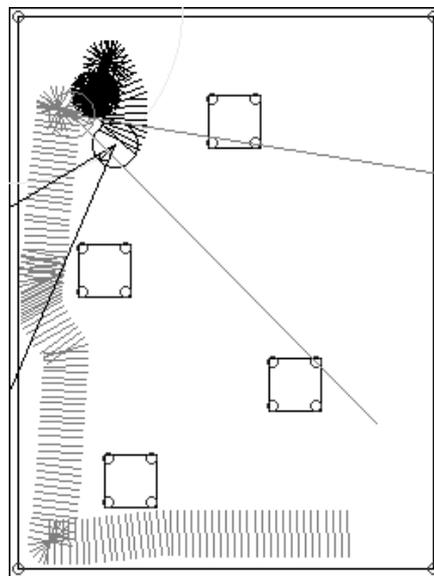


Figure 8. Wall Following approach. When predator is between wall and obstacle, it slides colliding with the wall

Regarding predator strategies, there is another problem: The prey robot is initially outside the sensory range of the predator. In fitness definition of experiment cases 1 and 3, where prey isn't

granted fitness for being in motion, prey can achieve fitness simply by staying immobile. And since predator's fitness, in case 1 doesn't include any navigation elements, it is likely that bootstrapping problems will appear as in early generations the fitness of the predator remains 0.

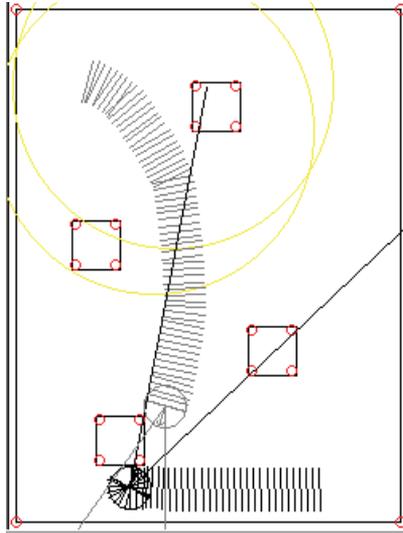


Figure 9. Prey hiding behind the obstacle. When predator tries to approach, stumbles in the upper corner of the obstacle

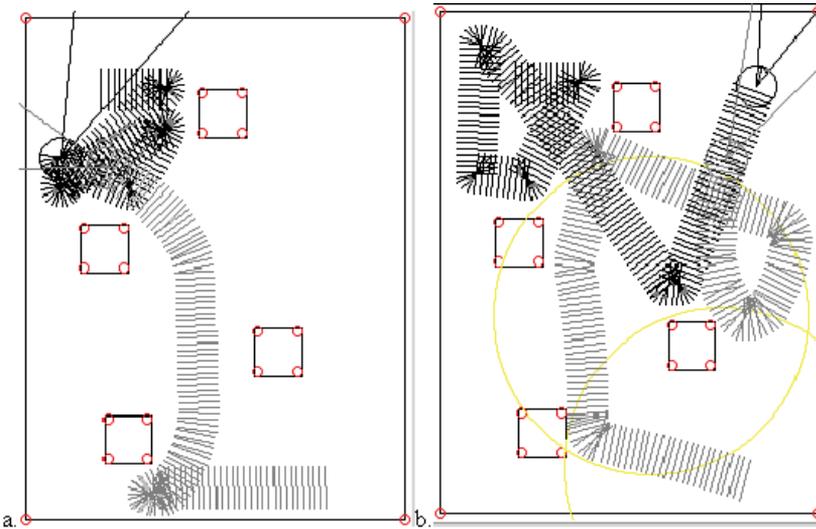


Figure 10. Conventional pursuit - evasion tactics. Contrary to previous examples, predator is based more on the rod sensor to locate the prey than to landmarks and ambient light readings

The evolutionary procedure has however found ways not only to produce obstacle avoiding strategies but also to use the obstacles in various ways. Figure 7 shows how can obstacles be used as landmarks to guide the predator near the prey: As prey wanders in the area confined by the two square objects, predator uses obstacles as landmarks to be guided in the

prey chamber. Figure 8 shows a variation of this strategy, where predator follows the outer wall of the arena to encircle the prey.

Another interesting strategy that utilizes obstacles is adopted by the prey: When fitness doesn't explicitly prevent contact with an obstacle, it is quite common to see the prey vehicle collide in the lower corner of an obstacle and monitor the above free area. In this way, the predator robot cannot approach successfully. Figure 9 shows this form of hiding. It is also the only tactic that can prove effective against a predator that has developed collision free navigation. Another parallax is adopted by the predator that sticks to an obstacle's corner waiting for the prey to pass near by. Conventional strategies like seeking or pursuit are shown in figures 10a and 10b.

6.2 Poverty of stimulus

Rod sensor readings are confusing for both agents as they indicate the position of the opponent but doesn't indicate if there are obstacles in the way. It has been mentioned above that this can cause agent to consider collision with opponent while they have collided with and obstacle.

Introducing the ambient light readings changes totally the way agents react. In all variations with ambient light readings, the agents use the rod sensor less, while depending on arena landmarks, internal dynamics and light readings to navigate upon or away the opponent. Also, using ambient light allows the predator to have more options in the "strike" phase of pursuit. It has been observed in cases where predator contacts the prey back-to-back, that using light reading it rotates on spot and strikes the prey. If light sensors are not used, in this case the predator navigated away from the prey.

It has also been observed that the modular neural network architecture evolution produces more awkward behaviours for both agents. This is a common problem in evolutionary robotics: When a certain architecture can solve a problem, adding more elements in the phenotype produces a larger genotype space (Harvey, 1997) that is harder for the evolutionary algorithm to optimize.

6.3 Measuring and evaluating

Observation of the produced behaviours is the most interesting part of an evolutionary experiment and the most revealing concerning the ability to evolve behaviours and solve problems with minimal resources (naive neural architectures, poor and noisy sensory inputs such as the IR sensory inputs). Yet, there is always need to have qualitative metrics of an experiment to be able to monitor the emerging evolutionary dynamics.

Measuring co-evolutionary experiments is, as mentioned before, more difficult than measuring static evolutionary ones. Instantaneous fitness cannot show much about the progress of the experiment, as evolving sophisticated behaviours on one species causes fitness drop on the other. Master fitness, on the other hand is the common denominator of all behaviours, but can be very low (compared to instantaneous fitness levels) when a specific strategy that can beat all opponent strategies cannot be found. Figure 11 shows instantaneous fitness variation for predator and prey in experimental setup 1 (simple prey fitness function form).

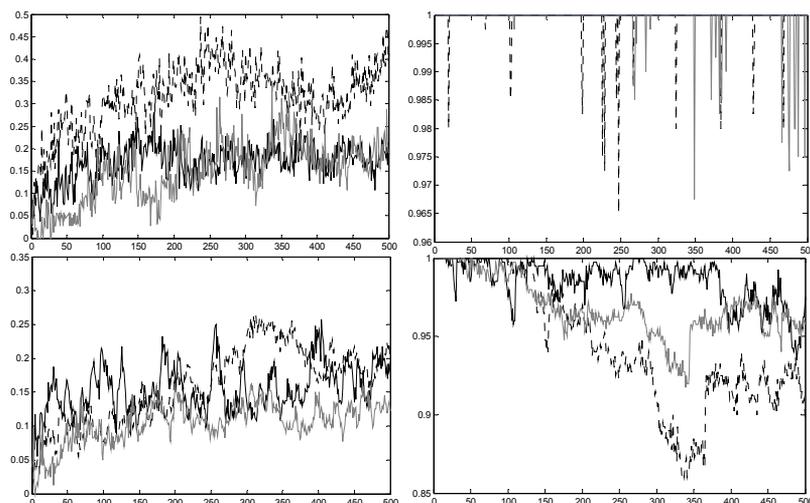


Figure 11. Instantaneous fitness function variation for predator (left hand) and prey (right hand) against number of generations for experiment setup 1 . The top plots are from tournament depth 1 experiments and bottom from tournament depth 10. Black: setup A (simple NN) Dotted: setup B (simple NN, ambient light input) and grey: setup C (Modular architecture, ambient light). All plots are average of 4 experiments with same setup

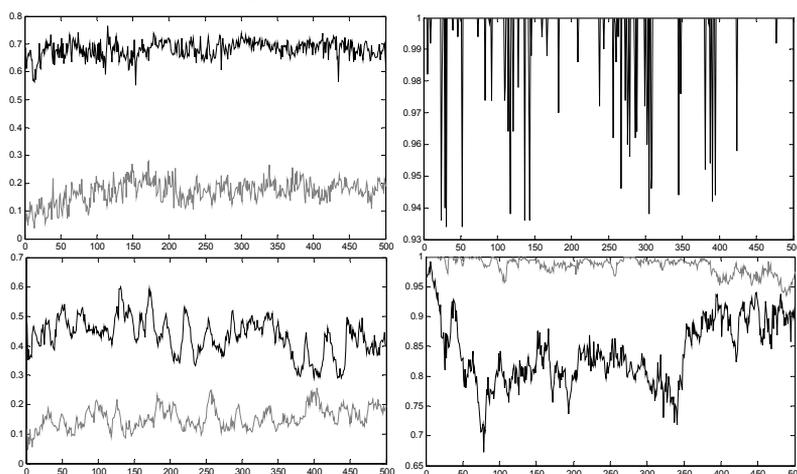


Figure 12. Instantaneous fitness for predator (left part) and prey (right part) in arena with obstacles (grey) and arena without obstacles (black) for experimental setup 1A. Upper Part: tournament depth 1. Lower: Tournament depth 10

Instantaneous fitness shows that the environment is totally favourable for the prey, as best individual's instantaneous fitness seldom is lower that 0.95, especially in tournament 1 depth. Figure 12 shows the instantaneous fitness variation in experiment 1A in an arena with obstacles and another without obstacles.

While the comparison in figure 12 shows a distinctive advantage for the prey robot, the waveforms are similar which seems that the same dynamics that can be monitored in a simple predator-prey setup (no obstacles) can be found when obstacle avoidance must also emerge.

Figure 11 partially proves the poverty of stimulus hypothesis as the predator that used the ambient light sensor gathers more fitness. It also seems that the modular architecture didn't cope well with the problem. The reason is that since the gene space is larger, it takes more generations and bigger population sizes to cope with the problem at hand.

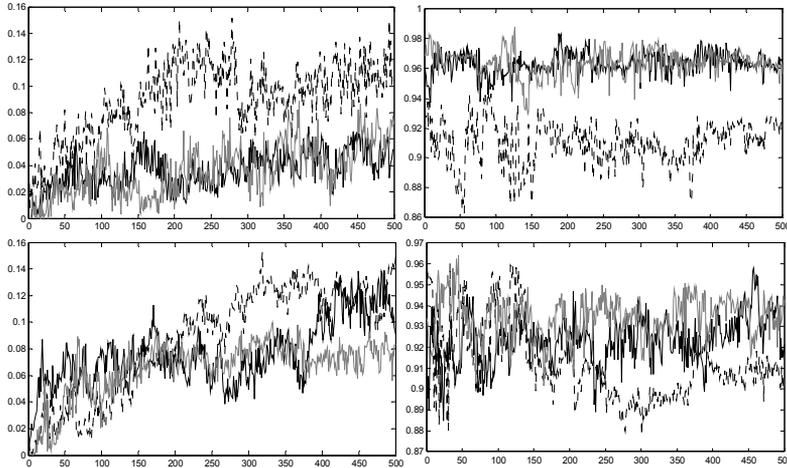


Figure 13. Master fitness comparison for plain fitness function definition using different neural setups (experimental setups 1A-1C). Left side: predator – right side prey, upper part : tournament Depth 1, lower part : tournament Depth 10. Black: setup A (simple NN) Dotted: setup B (simple NN, ambient light input) and grey: setup C (Modular architecture, ambient light). All plots are average of 4 experiments with same setup. Values are average of 4 experimental runs

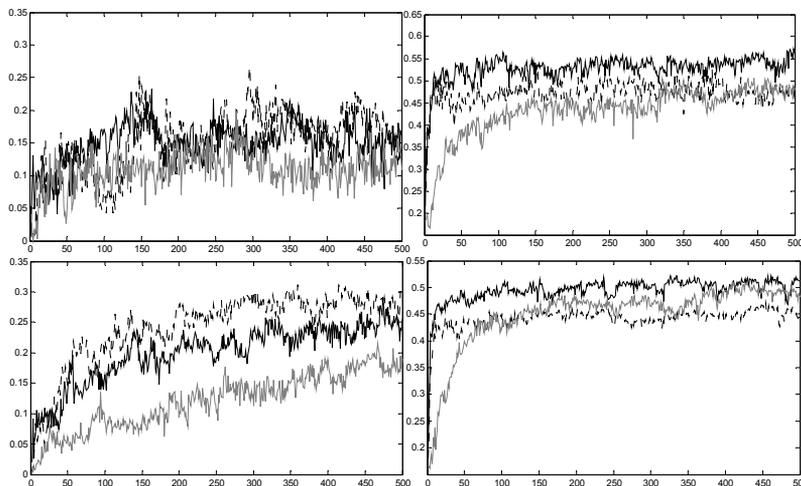


Figure 14. Master fitness comparison for fitness function definition 2. Plot specific details same as figure 13. Values are average of 4 experimental runs

Master fitness function comparison can show whether this is a general case for the co-evolutionary experiment or simply the co-evolutionary dynamics have changed. Figure 13 shows the comparison for experimental setup 1 and figure 14 for experimental setup 2.

Figures 13 and 14 don't show a clear supremacy of setup B (plain NN – ambient light sensors) by comparing predator's master fitness in each case. In prey's master fitness we see that the corresponding deterioration is clearer, which means that this particular setup is more favourable for the predator agent. Modular architecture seems to have the worst score in both agents.

Figure 14 shows also that predator's fitness is higher in general when prey's fitness incorporates the obstacle avoidance element. The explanation for this is a common problem of evolutionary robotics: When separate behaviours must evolve in parallel in an agent, evolution can paralyze in early generations by lack of useful lifetime experiences. In garbage collecting Khepera Nolfi (1997) and Ziemke et al. (1999) faced this problem since in early generations the Khepera agent didn't encounter objects in order to evolve the gripper handling behaviour. The solution to that was to program the simulator to put an object in front of the agent in the first generations.

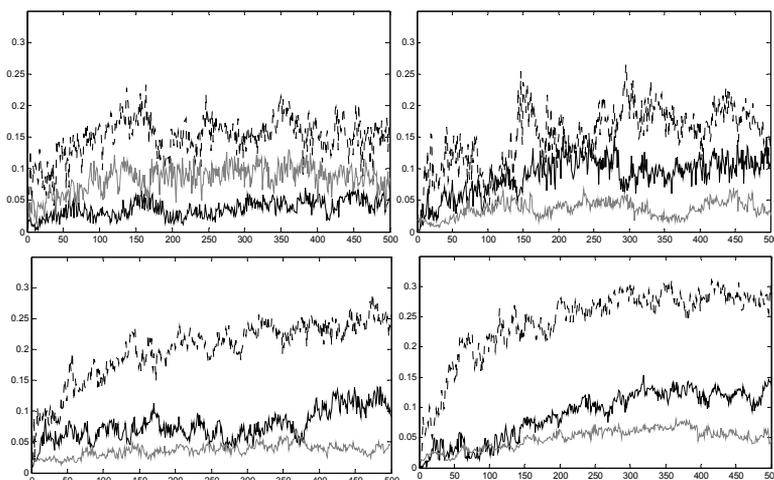


Figure 15. Comparison of the predator's master fitness for the 3 fitness function definitions setups left: Plane NN architecture, right : Plain with ambient light. Top : tournament 1 depth, bottom: tournament 10 depth. Black: fitness definition 1 (plain prey fitness), dashed: fitness definition 2, grey: Fitness definition 3 (plain with defined start generation)

In the previous section we saw that when prey's fitness doesn't favour obstacle avoidance and navigation in the arena, the favourable strategy for the prey was to stay immobile in the initial location or rotate on spot. This way it evaded being detected by predator agent's rod sensor. By forcing the prey to move around, useful rod sensory reading are generated for the predator causing master (and instantaneous) fitness to rise. The ascending trend in both predator's and prey's master fitness can be explained by the emergence of more refined obstacle avoidance.

Figure 15 seeks to give an answer to the initial question of the experiments presented in this chapter, which incremental evolution method is the best for the predator-prey task in the arena with obstacles. By comparing the master fitness variation of all setups (excluding the modular architecture which was proved inappropriate for the task) it seems that regarding predator emerging behaviours, fitness definition 2 gives the best results.

7. Conclusions

The possibility to provide an experimental framework for evolutionary biology and evolutionary game theory are the main strengths of the coevolutionary methodology. E.g.,

Cliff and Miller (1995a) rationalized the usability of co-evolutionary experiments with robotic agents in order to explain natural phenomena such as the emergence of protean behaviours in animals that usually are prey to others. By conducting experiments with simulated robots they were able to reproduce the phenomenon as it progresses and proceed with a game-theoretical analysis. As can be seen by the large number of corresponding publications in artificial life and evolutionary biology, evolutionary robotics greatly interacts with these areas.

The question whether coevolutionary methodology is capable of providing better robotic controllers than conventional evolutionary methods is quite hard to answer. First and foremost, the motivation behind coevolutionary experimentation is more to mimic biological procedures than produce competitive robotic behaviours. However, even when there are no optimistic results, co-evolution has the methodological advantage of being open-ended by achieving gradual complexification of the competing agents.

The experiments presented in this chapter proved that combination of behaviours can be done in competitive coevolutionary situations. The results (for the predator agent) were poor, something that can be explained by the fact that the two evolving strategies were controversial: One sought maximization of IR readings at some point (when contact with prey was made) while the other sought minimization of IR readings (collision avoidance). The predator's fitness has been extensively used as it shown variation across generations, something that was not the case for prey's fitness, which was high due to environmental advantage. Adding more sensory input improved the performance of the predator and the coevolutionary dynamics in general, while the hand made modular architecture proved poorer than the simple recurrent network.

The architectural evolution of neural controllers is something that must be further investigated: the experiments have shown that the architectures shown were under qualified to cope with the complexity of the combined tasks. However, in the experiments presented here, it was possible to see strategies like hiding or lurking emerge. In such strategies, the agent collided with an obstacle and then escaped by changing the direction of movement. Similarly, in many cases it was possible to observe the obstacles being used as landmarks to help the predator navigate towards the prey's initial location. Using landmarks, the predator didn't need to use the rod sensor except for the final phase before contacting the prey. This way the rod sensor reading that were confusing (as seen in section 4.2) when the prey was behind an obstacle were negated.

8. Future Work

This line of experiments has left open the point of what could be an optimal network architecture for problems of increasing complexity. Several researchers have conducted experiments in which the neural architecture was subject to change, either in genotypic (variable length GAs) (Harvey, 2001), (Husbands et al., 1998), or phenotypic (by using some developmental process during lifetime) level (Michel, 1997). Also, the ability to train the neural network during lifetime use delta or Hebb rule must be investigated.

Apart from the robotic controller internals, it is of great interest to study co-evolutionary experiments that include more than two species of species teams, clonal or aclonal. Experiments in these areas have shown the emergence of communication with or without dedicated channels (Quinn, 2001) or stigmergic collaboration (Caamano et al., 2007). By combining this feature with the multi-objective nature of multi-team situations, many interesting features can be studied, especially in the behavioural level.

9. References

- Brooks, R.A. (1999). *Cambrian Intelligence*, MIT Press, ISBN 0262522632, Cambridge, MA
- Buason G. & Ziemke T. (2003). Explorations of Task-Dependent Visual Morphologies in Competitive Co- Evolutionary Experiments, In: W. Banzhaf et al. (Eds.) *Advances in Artificial Life - The 7th European Conference on Artificial Life* (pp. 763-770). Springer-Verlag, Berlin, Heidelberg
- Caamano P, Beccera, J.A., Duro , R.J, Bell F. (2007). Incremental Evolution of Stigmergy-Based Multi Robot Controllers Through Utility Functions. *KES2007/WIRN 2007, Part II*, Springer-Verlag LNAI 4693, pp. 1187-1195, ISBN 1889994606
- Carlsson, J. & Ziemke, T. (2001). YAKS - Yet Another Khepera Simulator, In: Rückert, Sitte & Witkowski (eds.), *Autonomous minirobots for research and entertainment* (pp. 235-241), Paderborn, Germany: HNI-Verlagsschriftenreihe, ISBN 3935433069
- Christensen A.L. , Dorigo M. (2006). Incremental Evolution of Robot Controllers for a Highly Integrated Task, S. Nolfi et al. (Eds.): *SAB 2006*, LNAI 4095, pp. 473-484. Springer-Verlag, ISBN 1595930108, Berlin Heidelberg.
- Cliff & Miller (1995a). Co-evolution of pursuit and evasion I: Biological and Game-theoretical foundations, *Technical Report CSRP311*, University of Sussex, School of Cognitive and Computing Science
- Cliff & Miller (1995b). Tracking the red Queen: Measurements of adaptive progress in coevolutionary simulations, In: Moran, F., Moreno, A., Merelo, J. J., and Chacon, P. (eds.) *Advances in Artificial Life, 3rd European Conference on Artificial Life*, pp. 200-218, ISBN 978-3540594963
- Cliff & Miller (1995c). Co-evolution of pursuit and evasion II: Simulation methods and Results, In: *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior* , pp. 506-514. MIT Press, Cambridge, USA.
- Dawkins, Richard (1996). *The Blind Watchmaker*. New York: W. W. Norton & Company, Inc.. ISBN 0-393-31570-3.
- Floreano D. & Nolfi S. (1997). God save the red Queen: Competition in evolutionary robotics, In J.Koza et al. (Eds.), *2nd Conference on Genetic Programming*, San Mateo, CA: Morgan Kaufmann, pp.398-406.
- Floreano D., Nolfi S., Mondada F.(2001). Co-evolution and ontogenetic change in Competing Robots, In M. Patel, V. Honavar, and K. Balakrishnan (eds.), *Advances in the Evolutionary Synthesis of Intelligent Agents*, Cambridge (MA): MIT Press, pp. 273-306, ISBN 978-0262162012
- Gomez F, Miikkulainen R. (1997). Incremental evolution of complex general behavior. *Adaptive Behavior*, 5(3-4), pp. 317-342
- Harvey I. (1997) Artificial Evolution for Real Problems, *5th Intl. Symposium on Evolutionary Robotics*, Tokyo April 1997. Invited paper. In: *Evolutionary Robotics: From Intelligent Robots to Artificial Life (ER'97)*, T. Gomi (ed.). AAI Books
- Harvey I. (2001). Artificial evolution :A continuing saga. In Takashi Gomi, editor, *Proc. of 8th Intl. Symposium on Evolutionary Robotics (ER2001)*. Springer-Verlag LNCS 2219, pp. 94-109.
- Haynes T. and Sen S. (1996). Evolving behavioral strategies in predators and prey. In Gerhard Weißand Sandip Sen, editors, *Adaptation and Learning in Multiagent Systems*, pp. 113-126. Springer Verlag, LNAI 1042, Berlin, ISBN 3540609237
- Husbands, P. and Smith, T. and Jakobi, N. and O'Shea, M. (1998). Better Living Through Chemistry: Evolving GasNets for Robot Control, *Connection Science*, 10(4), pp185-210

- Jakobi N., Husbands P., and Harvey I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. In F. Moran, A. Moreno, J. Merelo, and P. Chacon, editors, *Advances in Artificial Life: Proc. 3rd European Conference on Artificial Life*, pp. 704--720. Springer-Verlag, Lecture Notes in Artificial intelligence 929
- K-Team, S. A. (1995). Khepera: User Manual. Switzerland <http://ftp.k-team.com/khepera/documentation/KheperaUserManual.pdf>
- Mermigkis I, Petrou L. (2006). Exploring Coevolutionary Relations by Alterations in Fitness Function: Experiments with Simulated Robots, *Journal of Intelligent and Robotic Systems*, Volume 47, Number 3, November 2006, pp. 257-284.
- Michel O., Clergue M., Collard P. (1997). Artificial Neurogenesis: Applications to the Cart-Pole Problem and to an Autonomous Mobile Robot. *International Journal on Artificial Intelligence Tools* 6(4):pp. 613-634
- Miglino O., H. H. Lund, and S. Nolfi (1995). Evolving Mobile Robots in Simulated and Real Environments. *Artificial Life*, Vol. 2.4: pp. 417--434
- Nietschke G(2003). Emergence of Cooperation in a Pursuit-Evasion Game, *International joint conference on Artificial Intelligence*, 2003, vol. 18, pp. 639-646
- Nolfi S. & Floreano D (1998). Co-evolving predator and prey robots: Do 'arm races' arise in artificial evolution?, *Artificial Life*, 4(4), pp. 311-335
- Nolfi S. & Floreano D.(2000). *Evolutionary robotics*, Cambridge, MA: MIT Press, ISBN 978-0262140706
- Nolfi, S. (1997). Using Emergent Modularity to Develop Control Systems for Mobile Robots, *Adaptive Behavior* 5, 3-4, pp. 343-363
- Potter M., Meeden L. A., Schultz A. (2001). Heterogeneity in the coevolved behaviors of mobile robots: The emergence of specialists, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pp.1337-1343. Morgan Kaufmann
- Quinn M.(2001):Evolving Communication without dedicated communication channels, In J.Kelemen and P. Sosvk, editors, *ECAL01*, pp. 357--366. Prague: Springer.
- Rosin, C., Belew, R. (1996). New Methods for Competitive Co-evolution. *Evolutionary Computation* 5:1 pp.1-29
- Schultz, A. C., Grefenstette, J. J., and Adams, W. L. (1996), RoboShepherd: Learning a Complex Behavior. *Proc. of the Robots and Learning Workshop (RoboLearn '96)*, May 1996, Key West, Florida, pp. 105-113. NCARAI Report AIC-96-030
- Sims K. (1994). Evolving 3D Morphology and Behavior by Competition, *Artificial Life IV Proceedings*, ed.by Brooks & Maes, pp. 28-39, MIT Press, ISBN 0262521903
- Stanley K.O, Miikkulainen R. (2004). Competitive Coevolution through Evolutionary Complexification, *Journal of Artificial Intelligence Research* 21 pp. 63-100
- Togelius J. (2004).Evolution of a subsumption architecture neurocontroller. *Journal of Intelligent and Fuzzy Systems*, Vol. 15, 1 : pp.15--20
- Whitehead S.D and Ballard D.H (1991). Learning to perceive and act by trial and Error, *Machine Learning*, Vol. 7, 1 : pp. 45-83.
- Ziemke, T., Carlsson, J. and Bodén, M. (1999). An experimental comparison of weight evolution in neural control architectures for a 'garbage-collecting' Khepera robot, in Experiments with the Mini-Robot Khepera - *Proceedings of the 1st International Khepera Workshop*, Löffler, A., Mondada, F., and Rückert, U. (eds.), pp. 31 - 40. Paderborn, Germany. HNI.

Evolving Behavior Coordination for Mobile Robots using Distributed Finite-State Automata

Pavel Petrovič

*Department of Applied Informatics, Comenius University
Slovakia*

1. Introduction

This chapter describes a finite-state automata approach to Evolutionary Robotics (ER) (Petrovic, 2007). Most of the efforts in the field of ER, see (Nolfi and Floreano, 2000) for very representative examples concentrate around controllers with various neural architectures, mainly due to their high plasticity and adaptability. Despite these obvious advantages, the resulting controllers usually have monolithic form with information represented numerically across the whole networks, where it is difficult to understand and explain why particular actions are taken in particular situations, enumerate the possible types of behaviors the controllers may produce, and achieve modularity required in more complex tasks. These features may be needed when the field reaches the level of real-world applications, for example to automatically verify the safety. In addition, many robotics researchers have demonstrated and reasoned that modular controller architectures with simultaneously executing behavioral modules – often called Behavior-Based (BB) architectures, are favorable as compared to centralized and top-down architectures. In our quarter of the ER community, the ultimate goal of the efforts is to find useful methods for automatic programming of mobile robots performing non-trivial tasks. By non-trivial, we mean tasks for which manual controller design by an engineer is difficult, or applications where the engineer is not available at the time of controller design or adaptation. Modular neural networks are an interesting field of interest, however, the research results did not satisfy us in their ability to compensate for their shortcomings. We suggest to use a BB architecture and apply Evolutionary Computation (EC) to learn/design the coordination mechanism of the behaviors. Although it would be possible to use EC to evolve both the behaviors and the coordination, evolving simple behaviors have been addressed by many previous works, and thus this time, we concentrate at the coordination mechanisms of a set of pre-programmed (pre-evolved) behaviors. When searching for a suitable formalism for representing the coordination mechanisms, we found that the robot behavior can be veritably modeled by finite-state automata (FSA). We also found that EC have been previously successfully applied to automatic programming of state automata, a detailed overview is in the section 3. As explained in section 4, we have designed a set of experiments, which indicate that for tasks that share properties with BB coordination mechanisms, the finite state representations outperform more conventional GP-tree representations. With this background, we have performed experiments with evolving

behavior coordination based on FSA for less trivial tasks for mobile robots and we have shown feasibility of this approach as described in detail in section 5.

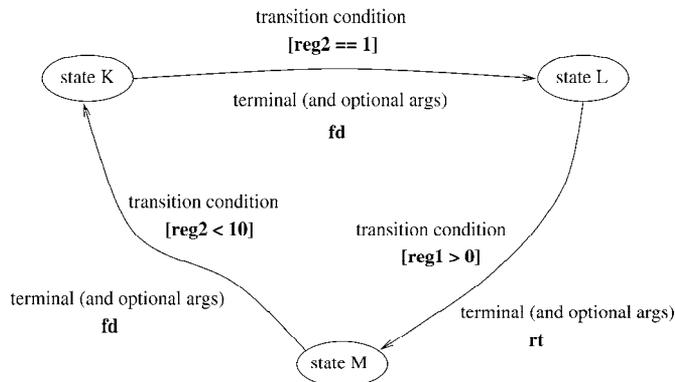


Figure 1. FSA representation. Transition conditions correspond to GP-tree non-terminals, however each transition also has an associated terminal. The state transitions can point to the originating state, and multiple transitions between the same two nodes, or with the same transition conditions are allowed (only one is executed when transition occurs)

2. Background and Motivation

There have been only few attempts to generate coordination mechanisms automatically based on the required task or robot purpose. For example, (Koza, 1992) evolves a robot controller based on Subsumption Architecture (SA) for wall-following task. He writes: “*The fact that it is possible to evolve a SA to solve a particular problem suggests that this approach to decomposing problems may be useful in building up solutions to difficult problems by aggregating task achieving behaviors until the problem is solved.*” Evolution of SA has been recently addressed by a thesis of (Togelius, 2003). The usefulness of these approaches can be supported by the following reasons:

- They are *innovating*: Automatic method might explore unforeseen solutions that would otherwise be omitted by standard engineering approaches used in manual or semi-automatic design performed by a human.
- They can provide robot controllers with *higher flexibility*: Mobile robots can be built for general purpose, and the coordination mechanism for different achievable tasks might have to be different, either for reasons of critical limits on their efficiency, the bounds on the controller capacity, or conflicting roles in different tasks. In such a case, generating the coordination mechanism based on task description might be required. Having the option of automatic coordination generation might save extensive amounts of work needed to hand-craft each coordination mechanism.
- They can *cope with complexity*: Manual coordination design might suffer from the lack of understanding of the real detailed interactions of the robot with its environment. These interactions might be difficult to describe analytically due to their complexity. Automatic coordination design might capture the undergoing characteristics of the robot interactions more reliably, efficiently and precisely.

A valuable inspiration for our work stems from the work of (Lee et al., 1998), where the more complex, high-level task is decomposed hierarchically and manually into several low-

level simple tasks, which can further be decomposed to lower-level tasks. The reactive controller consists of primitive behaviors at lowest level and behavior coordination at higher levels, both with the same architecture of interconnected logic-gate circuit networks. The evolution proceeds from the lowest level tasks up the hierarchy to the target complex task. The work has been continued on by the master thesis (Ostergaard, 2000), where a football player has been evolved with the use of co-evolution. The qualitative change is in the ability to work with internal state (as contrasted to purely-reactive controllers), and more complex architecture allowing two types of coordination: 1) a sequence of several states, and 2) selecting the module with the highest activation value; where the activation value is exponentially decreasing over time, and reset to maximum on request of the module.

3. FSA as Representation for Evolutionary Algorithms

FSA or FSM (we use these terms interchangeably) have been used as genotype representation in various works, although this representation lies on the outskirts of the evolutionary algorithms research and applications. Let us review the approaches first.

Evolutionary Programming, (Fogel, 1962, 1993) is a distinguished evolutionary approach that originally used FSA as the genotype representation. EP does not utilize recombination operators, and relies on mutations. The original EP works addressed the tasks of prediction, identification and control.

(Chellapilla and Czarnecki, 1999) introduce modular FSM subclass with restricted topology - in particular, the FSM are partitioned into several encapsulated sub-parts (sub-FSM), which can be entered exclusively through their starting states. The authors use modular FSM to evolve controllers for the artificial ant problem that was previously successfully solved by evolving binary-string encoded FSA in (Jefferson et al., 1992). They provide evidence that modular FSM perform better on this task than non-modular FSM, and they also provide evidence that direct encoding with structural mutations of non-modular FSM perform better than binary-string encoding used in (Jefferson et al., 1992). This idea of modular FSM has been adopted also by (Acras and Vergilio, 2004), who develop a universal framework for modular EP experiments and demonstrate its use on two examples.

(Angeline and Pollack, 1993) are experimenting with automatic modular emergence of FSA. They suggest to freeze and release parts of the FSA so that the frozen (or "compressed") parts cannot be affected by the evolutionary operators. The compression occurs randomly and due to the natural selection process, it is expected that those individuals where the compression occurs for the correctly evolved sub-modules will perform better and thus compression process interacts with the evolutionary process in mutually beneficial manner. Indeed, the authors document on the artificial ant problem that the runs with compression performed better than equivalent runs without compression. They reason: "*An explanation for these results is that the freezing process identifies and protects components that are important to the viability of the offspring. Subsequent mutations are forced to alter only less crucial components in the representation.*" (Lucas, 2003) is evolving finite-state transducers (FST), which are FSA that generate outputs, in particular, map strings in the input domain to strings in the output domain. FST for transforming 4-chain to 8-chain image chain codes were evolved in this work, while three different fitness measures for comparing generated strings were used: strict equality, hamming distance and edit distance.

An interesting piece of work by (Frey and Leugering, 2001) considers FSM as controllers for several 2D benchmark functions and the artificial ant problem. In their representation, the

whole transition function is represented as a single strongly typed GP-tree – i.e. a branching expression with conditions in the nodes that direct execution either to the left or to the right sub-tree, and finally arriving to a set of leaves that list the legal transition pairs (old state, new state).

In his PhD thesis, (Hsiao, 1997) is using evolved FSA to generate input sequences for digital circuits with the purpose of their verification, and fault detection. The author achieves best fault detection rate on various circuits (as compared to other approaches), except of those that require specific and often long sequences for fault activation.

(Horihan and Lu, 2004) are evolving FSM to accept words generated by a regular grammar. They use an incremental approach, where they first evolve FSM for simpler grammars, and gradually progress to more complex grammars. They use the term genetic inference to refer to their approach of generating such solution.

(Clelland and Newlands, 1994) are using EP with probabilistic FSA (PFSA) in order to identify regularities in the input data. The PFSA is a FSA, where the transitions are associated with probabilities as measured on input sequences. The EP is responsible for generating the topology of the FSA – number of states and how they are interconnected, and the transitions in PFSA are labeled according to their “fire rate”. This combination can be applied for rapid understanding of an internal structure of sequences.

(Ashlock et al., 2002) are evolving FSM to classify DNA primers as good and bad in simulated DNA amplification process. They evolve machines with 64 states in 600 generations. They used the weighted count of correct/incorrect classifications as their fitness function. However, they sum the classifications made in each state of FSM throughout its whole run. They argue that if only the classifications made in the final state were taken into account, the performance was poor. In addition, this allows the machine to produce weighted classification – how good/bad the classified primer is. The best of 100 resulting FSM had success rate of classification of about 70%. Hybridization, i.e. seeding 1/6th of a population of an extra evolutionary run with the best individuals from 100 previous evolutionary runs improved the result to about 77%. This work was continued in (Ashlock et al., 2004), where the FSM approach was compared to more conventional Interpolated Markov Models (IMMs), which outperformed FSM significantly.

In an inspiring study from AI Center of the Naval Research Laboratory, (Spears and Gordon, 2000) analyze evolution of navigational strategies for the game of competition for resources. The strategies are represented as FSM. Agent moves on a 2D grid while capturing the free cells. Another agent with a fixed, but stochastic strategy is capturing cells at the same time, and the game is over when there are no more cells to capture. Agents cannot leave their own territory. Authors find that the task is vulnerable to cyclic behavior that is ubiquitous in FSM, and therefore implement particular run-time checking to detect and avoid cycles. They experiment with the possibility to disable and again re-enable states (as contrasted to permanent and complete state deletion). They also compare evolution of machines with fixed number of states and evolution of machines, where the number of states changes throughout the evolutionary run. They discover that in the case of varying number, the machines utilize the lately-added states to lesser extent, as well as that deleting states is too dramatic for performance, and thus suggest merging or splitting states instead of deleting and creating states. Due to the stochastic algorithm of the opponent agent in the game of competition for resources, the fitness function must evaluate each individual in many different games (G). Authors disagree with others who claim that keeping G low can

be well compensated by higher number of generations and conclude that it results in unacceptable sampling error. The authors therefore evaluate all the individuals on fewer games (500), and if the individual should outperform the previous best individual, they re-evaluate it on many more (10000) games. Some further FSM-relevant references can be found for example in the EP sections in the GECCO and CEC conferences.

4. Analysis of FSA vs. GP-tree Representations

Each robot performing some activity is always in some state while it reactively responds with immediate actions or it proceeds to other states also as a response to environmental percepts – thus the activity of a robotic agent can be modeled by a state diagram accurately. We believe that state-diagram formalisms can in fact steer controllers themselves and be the back-bone of their internal architecture. Secondly, we believe that the state automata are easier to understand, analyze, and verify than other representations, for example neural networks. Thirdly, we believe that state automata are more amenable to incremental construction of the controller, because adding new functionality involves adding new states and transitions, and making relatively small changes to the previous states and transitions.

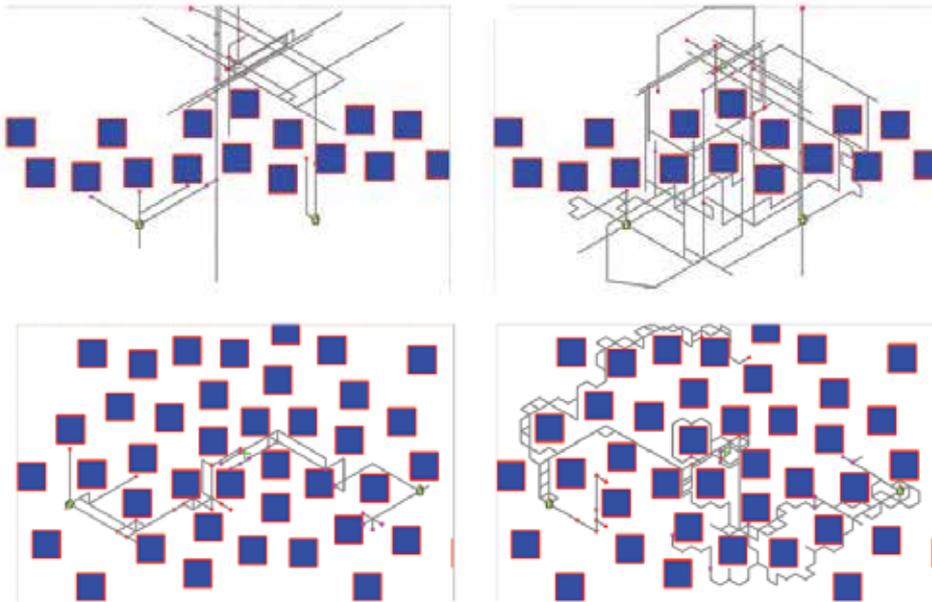


Figure 2. The best trajectories from all generations of one evolutionary run, two starting locations. GP-tree on the left, FSA on the right. The topology of the trajectories follows the topology of the representation: FSA are better at encoding loops and strategies, GP-trees are better at searching and extending the trajectories similar to the iterative deepening algorithm

In order to better understand the character of the FSA representation, we have devised a series of tasks and compared the performance of GP with program-tree representation against FSA representation, see Figure 1. In our implementation of GP-tree representation, the evolved programs are binary trees, with terminals (actions) in the leaves and non-

terminals (control structures *if*, *while*, *repeat*, *seq*) that utilize binary relations in the nodes. Both terminals and non-terminals can require arguments, which come in form of registers that can be mapped to sensor inputs or internal state variables. In our implementation of the FSA representation, the automaton consists of N states, which are connected by ordered set of state-transitions associated both with a transition condition and an action. The first transition with a satisfying condition is performed and an associated action is executed. For details of the formalisms and experiments, please refer to (Petrovic, 2007). In this section, we review the most interesting points.

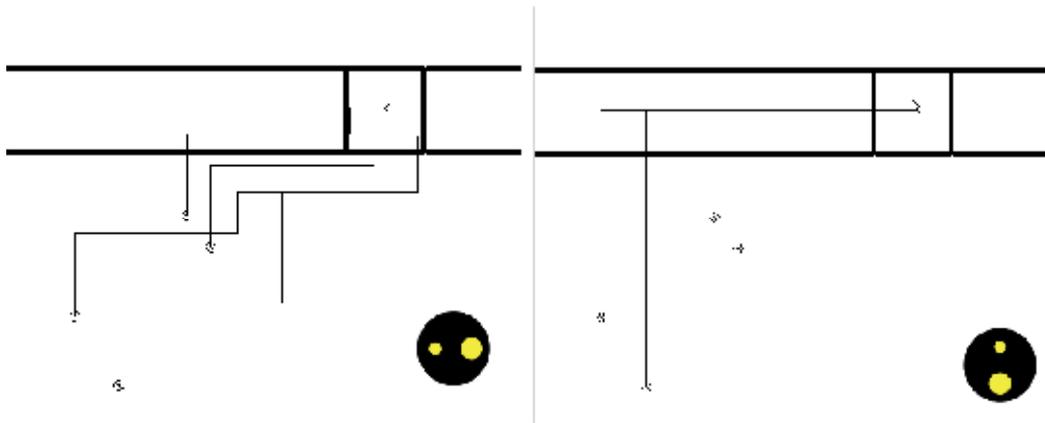


Figure 3. Trajectories for the evolved individuals in *selected* runs for both representations. The four starting locations are marked by a small cross in a circle and the final positions are marked by a small cross. Individuals in both representations utilize the light sensor, however, the FSA solution is cleaner – moving straight in the middle between the two horizontal lines, turning right and finding the target square. The GP-tree representation is approaching the target in a stair-like movement and faces difficulties to align with the target location correctly when the sensory experiences in the final segments of the trajectories vary. (The robot was moved to the bottom right just to disclose the trajectories.)

We have experimented with two simple 2D robot navigation tasks and several symbolic sequence tasks as follows: In the first task, robot was to navigate as close as possible to a specific target location (the distance determined the fitness of the solution), while avoiding collisions with obstacles (collisions subtracted from the fitness), see Figure 2. A binary sensor indicated whether the robot is heading roughly in the direction of the target. Even though there was only slight difference in the performance of both representations, inspecting the actual strategies reveals that GP program trees are better at incrementally extending the trajectories segment by segment in an iterative search process, while the FSA solutions are effective at encoding strategies. For instance, the solutions evolved in earlier generations shown at bottom right of Figure 2 avoid the obstacles always from the left-hand side. That brings the robot quite far from the target, but still closer than the starting location, however, it is a strategy that can be further modified to a correct solution. FSA are better at encoding loops, because the state transitions allow easy creation of arbitrary loop structures, while the sequential depth-first execution of the program trees is very prohibitive. The topology of the trajectories evolved in the consecutive generations reflects the topology of the representation itself: notice the tree structure of the trajectories of the GP program trees

and the loop structures of the FSA. Similar effects were found in results from a docking task, where a robot was placed in a bottom-left quadrant of a 2D environment and was required to park into a square. While the GP program trees achieved slightly higher target accuracy on average, the best evolved FSA solutions have a cleaner strategy, see Figure 3, and Figure 4 right.

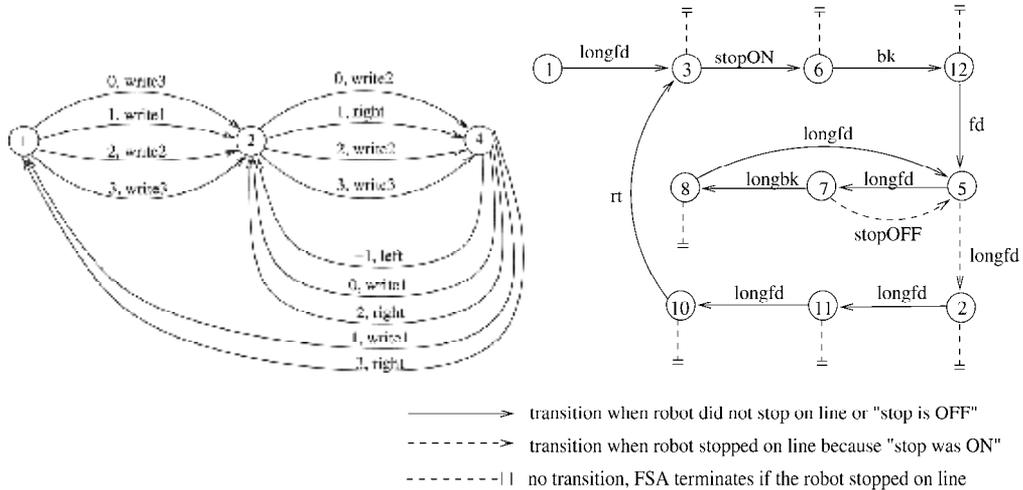


Figure 4. *On the left*: The best evolved FSA in the switch task with three symbols. *On the right*: FSA that evolved in the docking task. The robot first moves forward in the loop of states 5-7-8 until it arrives to line, then it moves forward three times (states 2-11-10), turns right, and proceeds again until it stops at line (the left line of the target square). Next, it moves into the square (states 2-11-10 again), turns right facing now down, and finally the FSA terminates when the robot attempts to move back in the state 6 when it encounters a line (top line of the target square as the robot is facing down)

In the symbolic task $(abcd)^n$, a tape machine was expected to replace a specified part of the input/output tape with repeating pattern of symbols $abcd$. We were surprised to find that GP program trees evolved solutions much faster, see figure 5. The following objective function was used:

$$\text{fitness} = 1 - s \cdot q_s - \sum_{i=1}^{n_{\text{starts}}} (e_i / (l_i \cdot n_{\text{starts}}) - r_i \cdot q_r) \quad (1)$$

Where e_i is the number of incorrect symbols in the output word (including extra placed or missing symbols) in the i^{th} input word, l_i is the number of symbols in the input word, n_{starts} is the number of random input words presented to the program, r_i is the number of execution steps, s is the size of the genotype, q_s and q_r are weight constants. Within 2000 generations, 74% of FSA runs evolved a correct solution, while 92% of GP-tree runs succeeded. The remaining 8% of GP-tree runs placed only 2 symbols incorrectly, while the incorrect FSA erred on 7-14 symbols. We found that GP performed so well due to its strong *while* non-terminal and the fixed structure of the loop required.

In the second symbolic task (labeled *switch*), the performance of FSA appeared to clearly outperform the GP program trees. In this task, the tape machine is to replace zeros in the

input string with a closest non-zero symbol found on the tape in the direction to the left. For example, the sequence

100040300002000130040000000003000020

should be transformed into:

1111443333322221333444444444333322

with the same objective function as in the previous task. In this case, the evolution of GP program trees was much slower than FSA, see Figure 6.

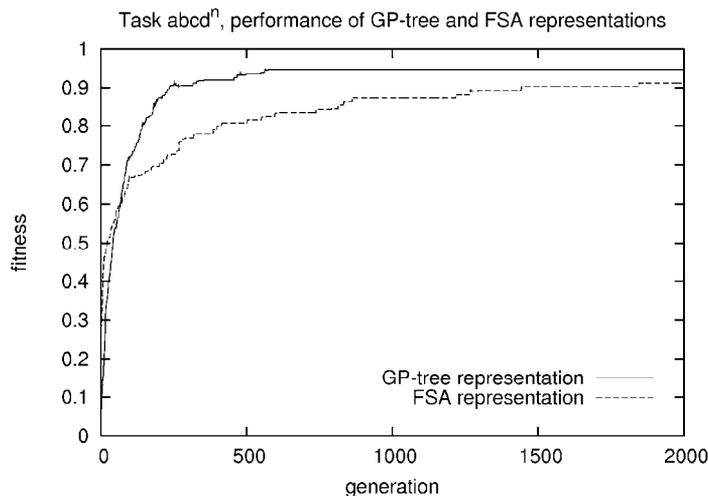


Figure 5. Performance of the FSA and GP-tree representations on the task $(abcd)^n$. Average of 25 (GP-tree) and 23 (FSA) runs. The programs were presented with input word containing 32 ones, and had 150 execution steps for writing the output word. Population size 300, prob. crossover 0.6, prob. mutation 0.7, brooding crossover (number of non-strict broods 3, 30% training samples used for brooding), combining crossover (GP-trees): 0.25, 15 strict-elite individuals, tournament selection (size 4, probability 0.8), max. GP-tree depth: 12, max. number of FSA states/transitions per state: 22/10, FSA shuffle mutation: 0.4

The task was a difficult one, and only 3 out of 10 FSA runs found a solution, therefore we ran experiments with a simpler version of the same task containing only three types of non-zero symbols. Within 2000 generations, all runs with the FSA representation evolved a correct solution, while no runs with the GP-tree representation found a correct solution, see Figure 7 and Figure 4 left for the smallest evolved FSA after pruning redundant transitions and state. The task can be interpreted from the robotic point of view as follows: the robot is always in one of the four (or three) states - it performs an activity of writing a specific symbol and navigating right on the tape. When a specific event in the environment occurs, it switches to a different state, performing a different activity (writing a different symbol). Whatever the current activity, it is capable to switch to any other possible state in a reactive manner to provide a required response. This is exactly the kind of interaction that is typical for robotic tasks, and behavior coordination, where the states of the behavioral modules switch according to the task and environmental context.

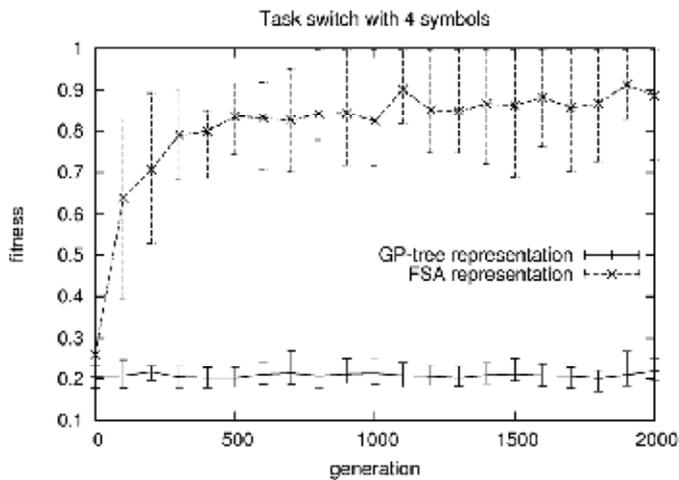


Figure 6. Average of the best fitness from 14 (GP) or 10 (FSA) runs of the switch task with four symbols. Population size 300, probability of crossover 0.5, crossover brooding of size 3, with 30% test cases used to evaluate brooding individuals, probability of mutation 0.9, 15 elites, each individual evaluated on 10 random strings, uniformly random input word length 10-60, maximum 10 continuous 0-symbols, maximum number of GP-tree or FSA execution steps 300, FSA: pshuffle=0.4, number of states 1-15, number of transitions 1-15, pcomb=0.25, maximum tree depth=15, The number of evaluations is proportional to the generation number. The error bars show the range of fitness progress in all runs. Tournament selection (FSA), fitness-proportionate selection (GP-trees)

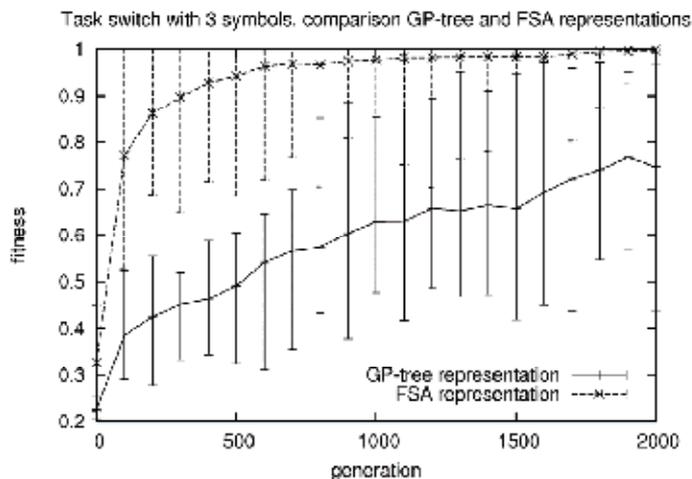


Figure 7. Average of the best fitness from 15 (GP) or 34 (FSA) runs of the switch task with three symbols, tournament selection, and other parameters as in the four-symbol experiment. The final fitness of all FSA runs outperforms the final fitness of the best GP-tree run

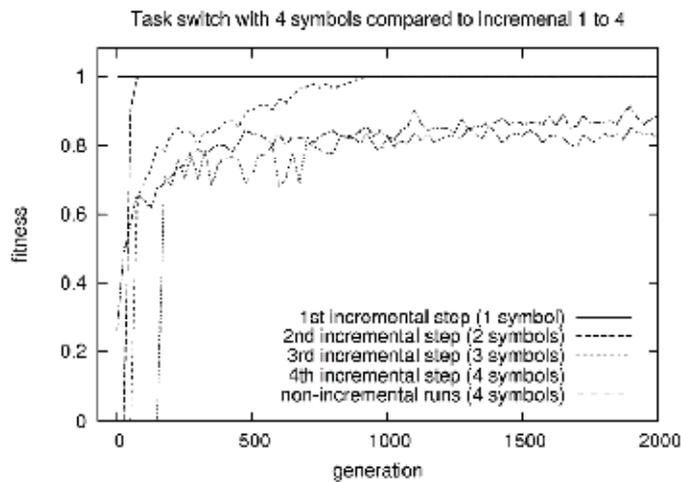


Figure 8. Average of the best fitness from 11 (incremental) and 10 (non-incremental) runs of a 4-symbol switch task, FSA representation and tournament selection, parameters as in Figure 7.

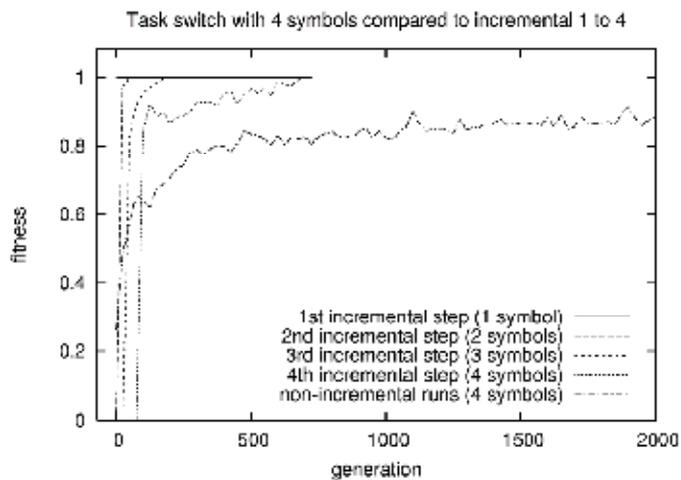


Figure 9. Average of the best fitness from 10 runs of the four-symbol switch task with FSA representation and tournament selection. Individuals were frozen at the end of each step. Other parameters remained the same

In further experiments with the switch task and FSA representation, we were interested to learn whether the evolution can find the solution faster, if supported by an incremental framework. When searching for a solution to the four-symbol task, we have first required the evolutionary algorithm to find a solution to the three-symbol version task and then introduced the input words containing the fourth symbol. A new action allowing to write symbol 4 was also added in the second incremental step. As shown at Figure 8, the performance of the evolutionary algorithm deteriorated in the case of the extra incremental step. Why is it that the incremental evolution failed in this case? At the first glance, the simplified three-symbol version of the task when preceding the fourth-symbol task should have made it easier to solve

the task, because it prevented the fourth symbol from appearing and thus did not require the state machine to react correctly to all the transitions from the states when writing the fourth symbol while evolving the other states. Finding a correct sub-solution to a three-symbol task should have been much faster, and that solution could have been easily extended to the four-symbol task as it is a subset. However, requiring the evolution of four-symbol solution to pass through a correct and complete three-symbol solution is a big burden! There are many possible paths from a random solution to a complete four-symbol solution, which do not pass through complete three-symbol solution. Requiring the evolution to pass through a particular point in the search space is a strong bias, which we call *incremental bias*. In cases when the advantages of the incremental setup outweigh the disadvantage of the incremental bias, the incremental evolution improves evolvability and convergence rate. Its application must be judged with respect to this general tradeoff principle.

We have attempted to construct a different scenario, which would be beneficial. Dividing the task to four steps with 1, 2, 3, and 4 symbols consecutively would have the same outcome as described above. However, freezing the best evolved automaton at the end of each incremental step, and letting the evolution in the further steps only to add new states and transitions simplifies the task greatly thus reaching the correct solution using the incremental scenario much faster as shown at Figure 9. Even higher speed-up was achieved when the terminal set in the later incremental steps was restricted, see (Petrovic, 2007).

The preliminary study covered in this section suggests that:

1. FSA share the structure with robotic tasks. They may be suitable for their modeling and implementation, especially when used to automatically program behavior coordination in BB controllers.
2. FSA and GP program-trees are two different representations, which both outperform each other on different tasks and are complementary in their features.
3. The incremental evolution can be beneficial for the performance of evolutionary algorithm only if its advantages outweigh the disadvantages resulting from the incremental bias of the evolutionary search.

5. Evolving Behavior Coordination

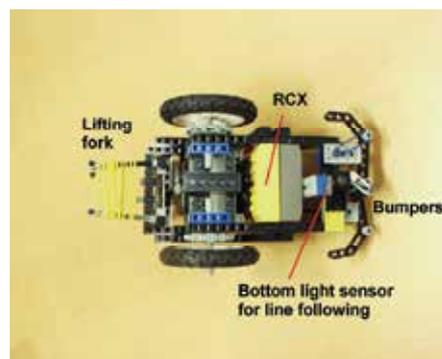
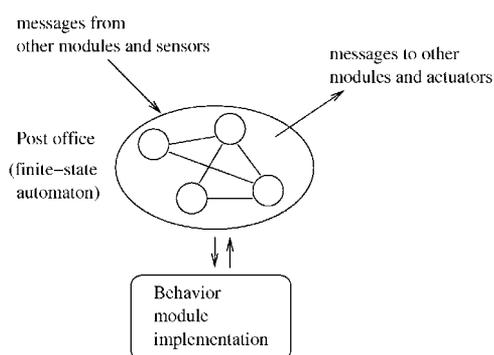


Figure 10. *On the left:* Single behavior module with its coordinating post-office finite-state automaton. *On the right:* bottom view of the robot. The upper light-sensor is installed on top side of the robot, and the lifting fork is operated by an extra motor and covered with rubber bands to increase friction

The goal of this work is to design controllers for mobile robots automatically by means of artificial evolution. We take the assumption that the hardware details of sensors and actuators are quite specific and the low level interactions of the controller with the robot hardware can be implemented efficiently and without much effort manually, before the target task is known: the behavior modules can be written in any available language or formalism manually. However, they can even be evolved automatically, if suitable. The part of the controller that we aim to design automatically here is the behavior coordination mechanism, in particular, a set of finite-state automata (FSA).

5.1 Controller Architecture and Evolutionary Algorithm

Let us explain the controller architecture on a simplified artificial example of a mouse robot acquiring a piece of cheese from a room and bringing it back to its mouse hole. The controller is shown at Figure 11. The mouse explores the room in random movements, and whenever it smells or sees the cheese, it moves in the direction of the smell, grasps the food, and drags it back to its home.

The core of the controller is formed by several modules, which are implementations of simple competencies (grayed boxes). These competences alone do not give the robot any purpose or any intelligent behavior yet. While performing their specific activity, they simply react to a predefined message interface and produce status messages whenever their actions or the incoming message of interest produce or indicate a significant outcome. For example, the “locating cheese” competence receives inputs from the vision and smelling sensor and produces a desired direction of movement, if the cheese is detected. Whenever the cheese is detected, it reports the event by an outgoing message. The competencies might be provided by the robot builders, or programmed in any programming language. Alternately, they can be hand-designed or evolved finite-state automata, GP program trees, or neural networks. The architecture does not limit their internal architecture. Most of the competencies have their own thread of execution. The competencies might be understood as an “operating system” of the robot that provides higher-level interface for controlling the low-level robot hardware. The intelligence and a particular purpose of the controller are encoded in a set of post-office modules, at most one post-office for each competence (post-offices are encircled by dashed boundaries at Figure 11). The post-office modules (Figure 10 left), are the communication interfaces of competencies with their peers and the remaining parts of the controller: sensors, and actuators. All messages received and sent by a particular competence module pass through its post-office module. The post-office modules in our architecture are finite-state machines, but other languages or formalisms could be used in place, as long as it is capable of filtering/transforming the incoming and outgoing messages of the module as needed for the specific robot task. Transitions can optionally result in generating new messages. In this way, the functionality of the competence module is turned on, or off, or regulated in a more advanced way, depending on the current state of the task, environment, and the robot performance represented by the state of the post-office FSA. The post-office simply filters or modifies the messages so that the competence module takes actions that are suitable in a particular situation. For example, the random turning competence will be activated only while the robot is exploring the room and searching for the cheese, or when it accidentally dropped and lost the cheese on its way back. The associated post-office module follows with the events performed by other modules that are

relevant, and adjusts its state to represent the current situation. Please refer to the section 5.3 below for another specific example.

We use the standard Genetic Algorithm (implemented using the GALib from MIT), with our specific initialization, crossover, and mutation operators. In the first stage, the designer prepares individual modules. For each module, he or she specifies the module message interface: the messages the module accepts and the messages it generates. In the second stage, the designer selects the modules for the controller and specifies lists of messages that can trigger incoming and outgoing transitions of the FSA associated with each module. The remaining work is performed by the evolutionary algorithm.

The genotype representation consists of blueprints of FSA for the set of modules for which the FSA are to be designed automatically (some modules might work without post-offices, other might use manually-designed post-offices, or some are held fixed because they are already evolved). An example of a genotype is in Figure 12. The number of states and the number of transitions in each state vary (within specified boundaries). Transitions are triggered by messages (incoming or outgoing) and have the following format:

(msg type, new state, msg to send out, [msg arguments], msg to send in, [msg arguments])

The GA-initialization operator generates random FSA that comply with the supplied specification. The crossover operator works on a single randomly selected FSA. It randomly divides states of the FSA from both parents into two pairs of subsets, and creates two new FSA by gluing the alternative parts together. It is designed so as to preserve as much information from both parents as possible, i.e. preserving and redirecting the state transitions consistently. For details, please refer to (Petrovic, 2007).

The mutation operator works upon a single FSA. One of these operations is performed (probabilities of mutations are parameters of the algorithm): 1) a random transition is created, 2) random transition is deleted, 3) a new state is created (with minimum incoming and outgoing random transitions); in addition, one new transition leading to this state from another state is randomly generated, 4) a random state is deleted as well as all its incident transitions, a random transition is modified: (one of its parts *new state, msg type, msg to send out, msg to send in* is replaced by an allowed random value), 5) a completely random individual is produced (this operator changes all FSA), 6) a random transaction is split in two and new state is created in the middle, 7) the initial state number is changed.

In our experiments, we use the roulette wheel and the tournament selection schemes combined with steady-state or standard GA with elitism. Other parameters of the algorithm include (with these default values): $p_{crossover}$ (0.3), $p_{mutation}$ (0.7), probabilities of mutations: $p_{new_random_transition}$ (0.25), $p_{delete_random_transition}$ (0.1), p_{new_state} (0.2), $p_{random_state_deleted}$ (0.05), $p_{random_transition_mutated}$ (0.25), $p_{new_random_individual}$ (0.05), $p_{split_transition}$ (0.05), $p_{change_starting_state}$ (0.05); $population_size$ (100), gen_number (60), $p_{population\ replace}$ (0.2), *number of modules in the controller* (10), specification of the message interfaces and trigger messages for all modules, initial and boundary values for *number of states and transitions, number of starting locations* for the robot for each evaluation, *timeout* for the robot evaluation run, specification of the *fitness function parameters*, input, output, and log file locations, please find more details in (Petrovic, 2007).

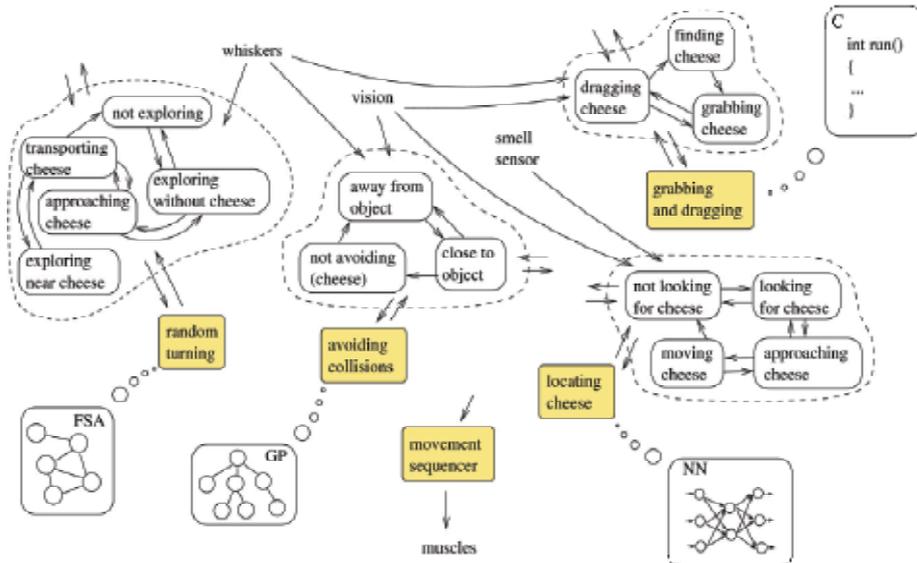


Figure 11. Example controller architecture for the task of a mouse acquiring cheese

5.2 Incrementality

Researchers observed in the past (Harvey, 1995) that evolving robot behavior is a hard challenge for any EA. The fitness landscape tends to be rough, and evaluation of each individual typically takes a long time. Trying to evolve more complex behaviors is often too difficult. Some groups, such as (Harvey, 1995, Lee et al., 1998) advocated the use of incremental evolution, where the complexity of the target task is decreased by decomposing it to several simpler tasks, which are easy enough to solve by an evolutionary algorithm. We have identified five different ways, in which an evolutionary robotic algorithm can be incremental:

Environment (where is the robot performing?): the earlier incremental steps can be run in a simplified environment, where the frequency and characteristics of percepts of all kinds can be adjusted to make it easier for the robot to perform the task. For instance, the number of obstacles or distance to the target can be reduced, the environment can be made more deterministic, the noise can be suppressed, landmarks can be made more visible, etc. An example of this type of incrementality is (Lund and Miglino, 1998), where box-shaped obstacles were replaced by more difficult U-shaped obstacles after the avoidance behavior for the former was evolved.

Task (what is the robot doing?): the earlier incremental steps can require only part of the target task to be completed, or the robot might be trained to perform an independent simple task, where it learns skills that will be needed to successfully perform in the following tasks. An example of this type of incrementality is (Harvey, 1995), where the gantry robot evolved forward movement first, followed by stages that required movement towards a large target, movement towards a small target, and distinguishing a triangle from square. For another example, we might first require a football playing robot to approach the ball, later we could require also to approach it from the right direction.

Controller (how is the robot doing it?): the architecture of the controller changes. For example, the final controller might contain many interacting modules, but the individual interactions can be evolved in independent steps, where only the relevant modules are enabled. In the later steps, the behavior might be further tuned to integrate with other modules of the controller. This type of evolutionary incrementality occurs seldom in the literature, but an example could be a finite-state machine-controlled robot negotiating a maze. The controller can be extended with a mapping module that is able to learn the maze topology, however, the output of the module has to be properly integrated with the output of the FSA. Non-evolutionary controller incrementality can certainly be seen in the Subsumption architecture and its flavors (Maes, 1990), and many later BB approaches. The task for the robot might require a complex controller, for example one with an internal state. Evolution can start with a simple controller that is sufficient for initial task and the controller can be extended later during the evolution. The change can be either quantitative, i.e. incrementing the number of nodes in a neural network, or qualitative, i.e. introducing a new set of primitives for a GP-evolved program.

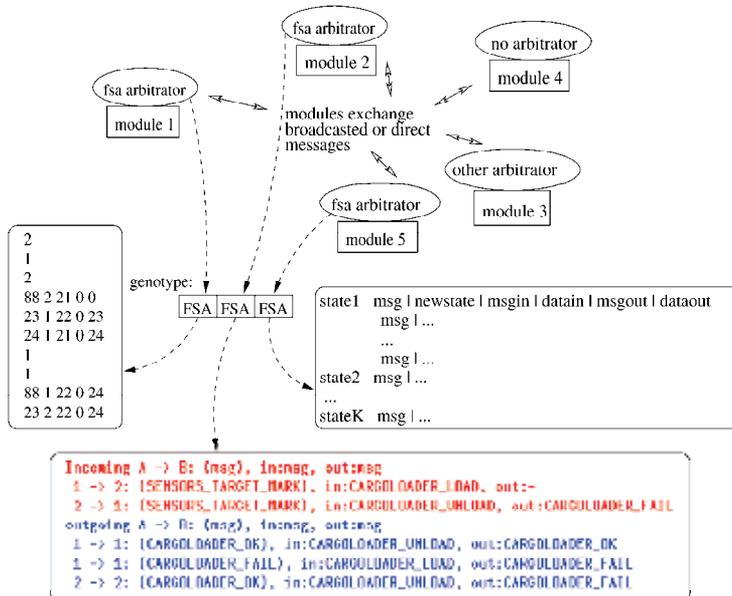


Figure 12. Controller architecture and genotype representation: left oval shows actual numeric genotype representation (it is a vector of numbers containing the number of states in FSA, number of incoming and outgoing transitions in each state, and detailed transition specifications), bottom oval shows a symbolic representation as viewed by a viewer utility (used to analyze the evolved post-offices), right oval shows the genotype structure for both incoming and outgoing messages for better explanation

Robot sensors/actuators (with what. . . ?) the dimensionality of the search space might be reduced by disabling some of the robot sensors and actuators before they are needed for the task evolved in each particular step. An example of this can be seen in Incremental Robot Shaping of (Urzelai et al., 1998), where the Khepera robot had first evolved the abilities of navigation, obstacle avoidance, and battery recharge, before a gripper was attached to it and the robot had to evolve an additional behavior of collecting objects and releasing them

outside of the arena. A subset of robot's equipment can be used in the early steps and more specific sensors and actuators added later.

Robot morphology (what form does the robot have?): the shape and size of the robot can be adjusted to make its performance better and reshaped according to final design in the later incremental steps. This kind of incrementality is also seldom seen in the literature. On the other hand, there are examples where the robot morphology itself is evolved (Lund, 2001). An example of morphological incrementality would be a vacuum-cleaning robot with the shape of an elliptical cylinder that needs to turn in proper direction to pass through narrow passages. It could be simplified to a circular cylinder to evolve basic navigation strategies and later updated to its final shape to achieve the proper target behavior. Another example is evolving the particular target shape layer by layer, if the physics allow.

From the implementation point of view, incrementality can be achieved by modifying the simulated environment, the objective function, the genotype representation and the corresponding controller implementation, and the configuration of the simulated robot. When evolving the target task in multiple steps, we do not necessarily require that the steps form a linear sequence. The behavior can be partitioned into simpler behaviors in various way, and the evolution progress will follow an incremental evolution scenario graph.

Another important issue is how to transfer a population from the end of one incremental step to another step. When entering a new step, already learned parts of the genotype can either remain frozen or continue to evolve. Sub-parts of the problem can be either independent, for example individual modules that can be tested separately, or depend to some extent on other parts – thus creating a dependence hierarchy (i.e. arranged in a tree, or in a graph as explained above). In case of dependencies, one can incrementally evolve the behaviors from the bottom of the hierarchy, adding upper layers and parallelize the algorithm later. This is a general framework. In order to find plausible solutions, EAs require that the initial population randomly samples the search space. However, the population at the end of one step is typically converged to a very narrow area, and thus it cannot be used directly as an initial population in the next step. We therefore propose to generate a new initial population from several ingredients:

- some portion of the original population containing the best individuals is copied,
- another part is filled with copied individuals that are mutated several times, and
- the remaining individuals are randomly generated.

However, it is also possible to blend the populations of two or more previous incremental steps. In principle, we propose that the evolutionary incremental algorithm will take for each incremental step a full specification of blending, copying, and mutation ratios for all genome sub-parts (such as finite-state automata) and all preceding incremental steps, see (Petrovic, 2007) for formal specification of the process.

For example, let us examine an incremental scenario with six incremental steps. The target genome in the last incremental step consists of three finite-state automata each corresponding to one of the behavioral modules that are subject to evolution, see Figure 13. In addition, there may be other modules in the controller, which are already designed and which are not evolved in the particular step – these are shown filled at the figure. In our example, the second incremental step continues to evolve the automaton corresponding to the first module, and begins with evolving the automaton corresponding to module II. The third and fifth steps are completely independent, and evolve the automata for module III, and for modules I and III, respectively. The fourth and the sixth steps merge populations

from multiple steps. Whereas the fourth step simply combines the automata for the module III coming from step 3, and automata for the modules I and II from the second step into common controller, the sixth step blends the populations from the fourth and fifth steps for both modules I and III, and further evolves the automata for the module II from the fourth step.

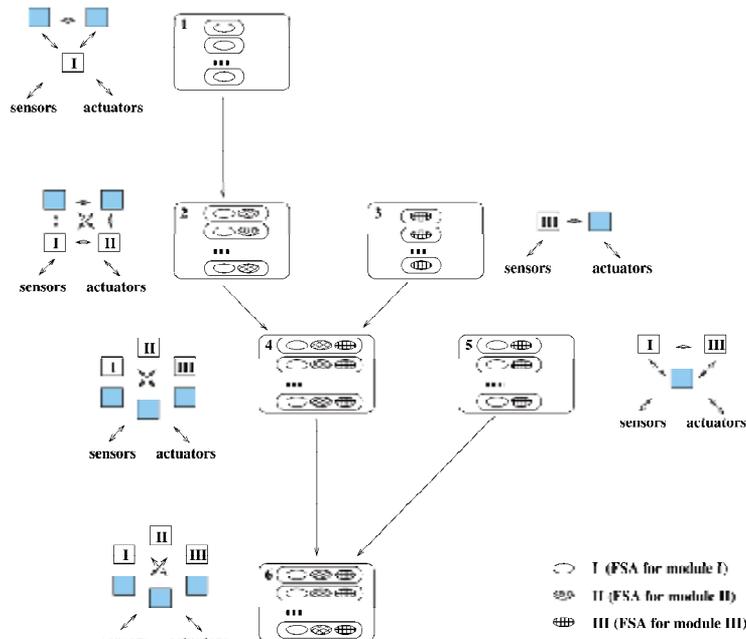


Figure 13. Population mixing in an incremental scenario

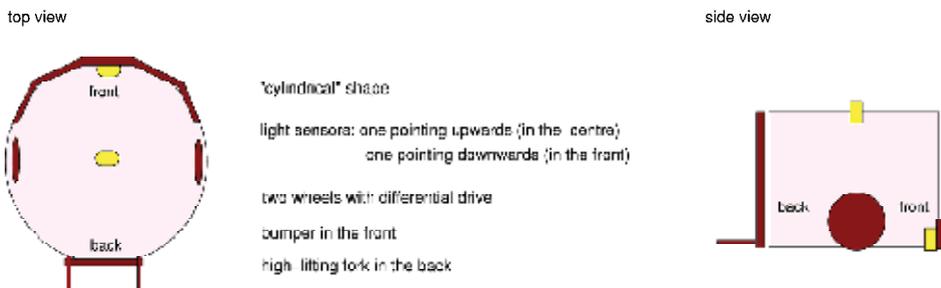


Figure 14. Simulated robot topology and features

5.3 Experiment

Our experiment has the goal to evaluate the controller architecture and the incremental evolution method proposed in the previous chapters and compare them to a manual design of the coordination mechanism in the controller. The RCX hardware platform offers high flexibility and a multitude of possible configurations for laboratory robotics experiments. We tested the implementation of our controller architecture on a high lifting fork robot built around a single RCX (Figure 10 right, Figure 14). The task for the robot is to locate a loading station, where the cargo has to be loaded, and then locate an unloading station to unload the

cargo, and repeat this sequence until the program is turned off. The robot exists in a closed rectangular arena with obstacles to be avoided. Both loading and unloading stations lie at the end of a line drawn on the floor. The start of the correct line to be followed at each moment is illuminated from above by light (an adjustable office lamp), Figure 15F. The light source located over a segment of the line leading to the loading station is automatically turned off when the robot loads the cargo, and it is turned on when the robot unloads the cargo at the correct location. The reverse is true for the light located over the line leading to the unloading station. Other lines might exist in the environment as well. We chose this task for four reasons: 1) compared to other evolutionary robotics experiments, it is a difficult task; 2) it is modular, in terms of the same behavior (finding and following line) being repeated two times, but with a different ending (either loading or unloading cargo), and therefore has a potential for reuse of the same code or parts of the controller; 3) it can be implemented both in real hardware and in our simulator (for the implementation of the switching lights, we used two standard office electric bulb lamps controlled by two X10 lamp modules and one X10 PC interface connected to a serial port of a computer that received an IR message from RCX brick when the robot loaded the cargo, which was in turn detected by an infrared emitter/detector from HiTechnic); 4) the task consists of multiple interactions, and behaviors, and thus is suitable for incremental evolution.

Our robot comes with a set of preprogrammed behavioral modules. We have coded them directly in the language C:

Sensors – translates the numeric sensory readings into events, such as robot passed over or left a line, entered or left an illuminated area, received an IR message from a cargo station, bounced into or avoided an obstacle.

Motor driver – accepts commands to power or idle the motors. The messages come asynchronously from various modules and with various priorities. The purpose of this module is to maintain the output according to the currently highest priority request, and fall back to lower priorities as needed. All motor control commands in this controller are by convention going through the motor driver.

Navigate – is a service module, which provides higher-level navigational commands – such as move forward, backward, turn left, as contrasted with low-level motor signals that adjust wheel velocities.

Avoidance – monitors the obstacle events, and avoids the obstacles when encountered.

Line-follower – follows the line, or stops following it when requested.

Explorer – navigates the robot to randomly explore the environment. It turns towards illuminated locations with higher probability.

Cargo-loader – executes a procedure of loading and unloading cargo on demand: when the robot arrives to the cargo loading station, it has to turn 180°, since the lifting fork is on the other side than the bumpers, then it moves the fork down, approaches the cargo, lifts it up, and leaves the station; at the unloading station, the robot turns, approaches the target cargo location, moves the fork down, backs up, and lifts the fork up again.

Console and Beep – are debugging purpose modules, which display a message on the LCD, and play sounds.

The coordination mechanism, which is our focus, consists of FSA post offices attached to individual modules. Figure 17 shows the hand-made FSA for the line-follower module. Other modules that use FSA are cargo-loader, avoidance, and explore. Figure 15 shows the six incremental steps and their respective environments for our main incremental scenario

(we refer to it as creative). Throughout the whole experiment, the robot morphology and the set of sensors and actuators remained unchanged. In the first three incremental steps, the task, the environment, and the controller were simplified. In the fourth and the fifth incremental steps, the task and the environment were simplified, but the controller already contained all its functionality. Evolution progressed to the next incremental step when an individual with a satisfactory fitness was found and the improvement ratio fell below a certain value, i.e. the evolution stopped generating better fit individuals. The improvement ratio m_n in generation n was computed using the following formula:

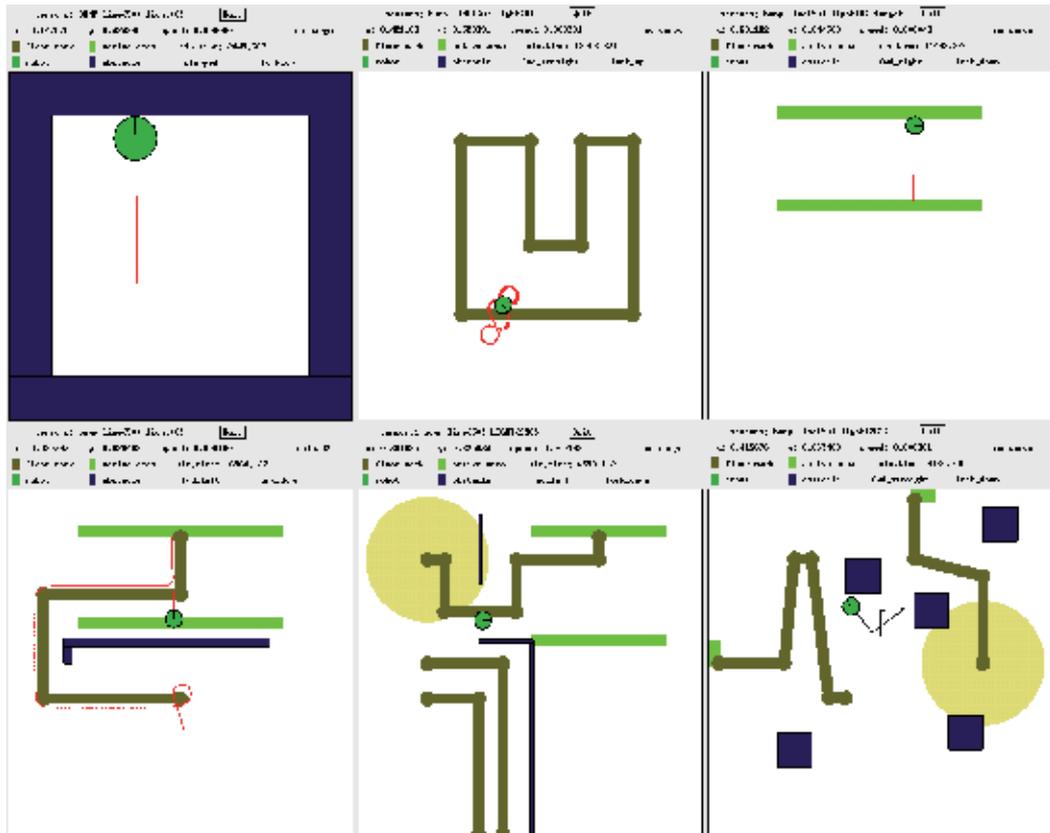


Figure 15. Experimental environments for 6 incremental evolutionary steps of creative incremental scenario, from left A – C, and D – F in the first and the second rows. A: *avoidance* – the robot is penalized for time it spends along the wall; B: *line following* – the robot is rewarded for the time it successfully follows the line; it must have contact with the line and should be moving forward; C: *cargo-loading* – robot is rewarded for loading and unloading cargo in an open area without lines or obstacles; D: *cargo-loading after line following* – follow-up of B and C, the robot is rewarded for loading and unloading cargo, but it has to successfully follow line to get to the open loading/unloading area; E: starting *line-following under light* – robot learns to start following the line that is under the light (it is started from different locations in order to make sure it is sensitive to light and not, for instance, to number of lines it needs to cross); F: *final task* – robot is rewarded for successfully loading and delivering the cargo, it uses the avoidance learned in A and behavior E

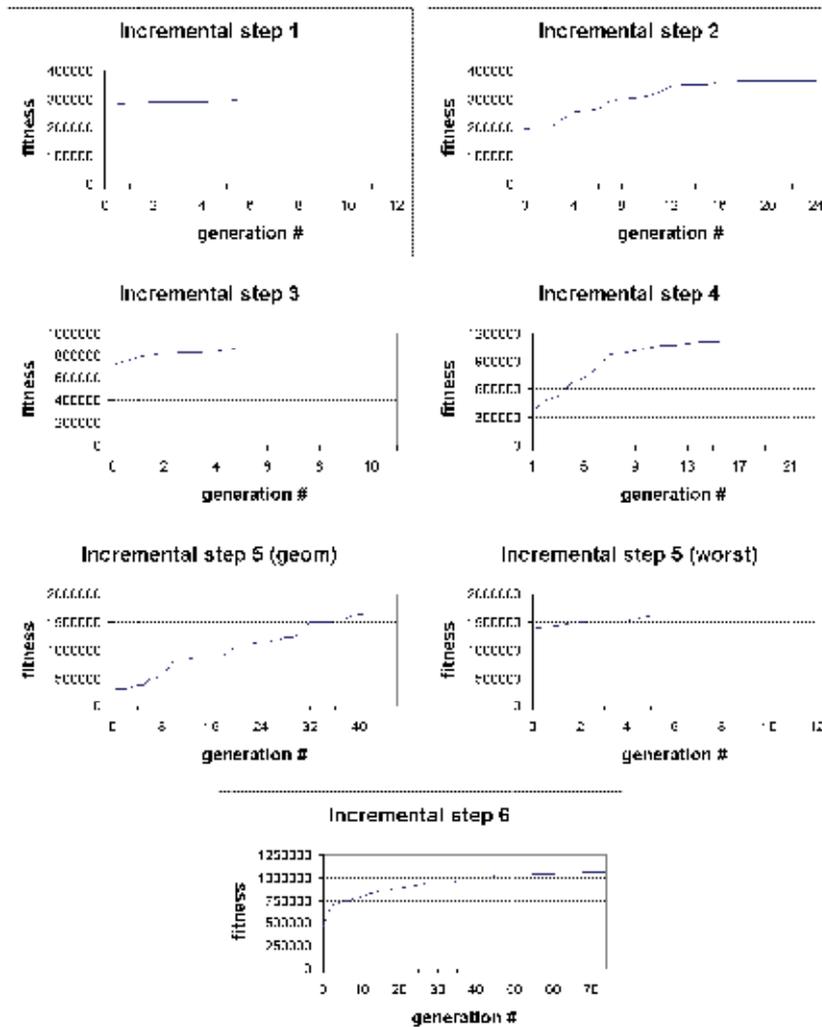


Figure 16. Best fitness for all incremental steps (averaged 10 runs), creative incremental scenario. In the incremental step 5 (geom), the assigned fitness is geometric mean of fitness achieved from the three different starting locations. This step was followed by 5 (worst), where the assigned fitness is the worst fitness achieved in the three starting locations

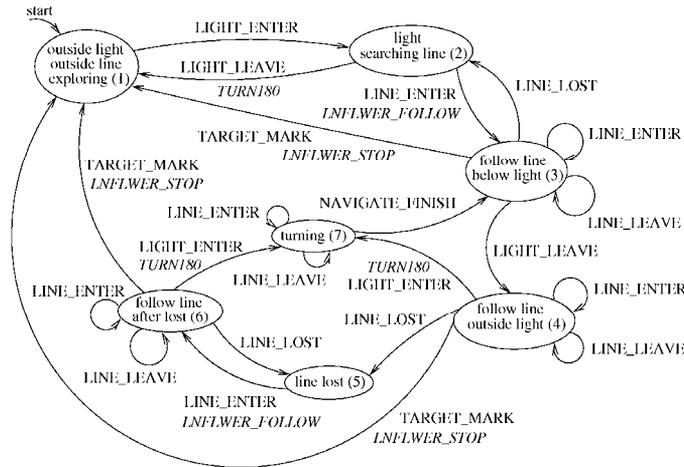


Figure17. Example of a manually designed FSA coordination for the line-follower module

$$m_n = \varphi \cdot m_{n-1} + (\text{best fitness}_n - \text{best fitness}_{n-1}) \quad (2)$$

where best fitness_i is the fitness of the best individual in population i , φ is a constant (we used $\varphi = 0.2$), and m_0 is initialized to $0.9 \cdot \text{best fitness}_0$.

5.4 Simulation

Simulations in ER are more complex than usual, because we simulate the behavior of a robot in its environment as contrasted to a simulation of some process that is an integral part of the simulated environment. The behavior of robot depends not only on the physical processes in the environment, but also on its own actions generated by its program – i.e. there are two simultaneous systems to be simulated. The simulation of the robot controller can be performed through direct emulation of the robot hardware/OS on the simulating hardware and OS. None of the existing simulators satisfied our needs, we chose to implement our own directly in language C, to be compatible with the compiling environment of the HW platform.

Our *lazy simulation* approach (inspired by lazy evaluation in functional programming languages) updates the state of the simulated system only when the robot controller interacts with the robot hardware. At that time instant, we suspend the emulated program, compute the current exact state of the simulated system (environment and robot) by mathematical formulas, and determine the outcome of the interaction that triggered the update. The temporal granularity is thus limited only by the CPU or bus frequency of the simulating machine. The emulated program is not interpreted by the simulator. It runs almost independently within the operating system of the simulating computer. Instead of accessing the robot hardware, it accesses the simulator that is waiting in the background. For example, the robot controller program might be computing without interacting with the robot hardware for some time, during which the robot crosses several lines on the floor, triggers switching of the light by entering an active area, passes below a light, and bounces to a wall, where it remains blocked for a while. At that point in time, the robot controller wants to read a value of its light sensor, for instance, and only at that point in time the simulator becomes active and computes the whole sequence of the previous events that

occurred, and the current location and situation of the robot and the environment. Finally, the required value of the sensor reading is determined and returned to the program, which resumes its execution. To achieve better performance, the simulator pre-computes expected events before resuming the simulated program. The pre-computed information helps to test quickly whether the state of the robot or environment has changed since the last “interrupt”, without processing all the data structures of the simulator. We call this approach lazy simulation because it uses maximum possible abstraction from the environment and performs simulation computation only when very necessary.

5.5 Results

To verify the controller architecture and task suitability, we have first designed the post-office coordination manually. This controller performed well and resulted in reliable cargo delivery behavior. We have also tested the controller on the real robot. The transition to the real-world settings was straightforward, except of the calibration of the sensors and timing of motoric actions. Still, in this experiment, the exact quantitative dimensions and distances played minor role, for the performance of the controller (except, perhaps, for the line-follower module), and therefore the distances and timings in the realistic actions did not need to correspond to the simulated one with 100% accuracy, and actual tuning of the timing could be performed separately for the simulated and realistic runs.

In the evolutionary experiments, we have tried to see if the evolutionary algorithm described above could evolve the target task by automatically designing all four FSA modules in a single evolutionary run. We ran the program for 20 times with a population of 200 individuals and 200 generations, and with a fitness function rewarding line-following, cargo-loading and unloading, distance traveling, and penalizing obstacles. However, none of the runs evolved the target behavior. To save computational effort, we have stored all previously evaluated genotypes with their fitness to the database. The objective function first checks if the genotype has already been evaluated and starts the simulator only in case of a new genotype. Furthermore, during the simulated run, we measure the fitness obtained by the best (or average of several best) individuals, and later, we automatically stop all individuals that achieved less than $q\%$ of the best measured fitness ($q = 5\%$) in one of the periodically occurring checkpoints, if this was feasible. All evaluated FSA and the trajectories of best-fitness improving runs were saved to files and extensive logs were produced for further analysis.

Later experiments followed the creative scenario with 6 incremental steps shown at Figure 15. The general fitness function used in this task had a form

$$\begin{aligned}
 f = & a + w_{\text{obstacle time}} \cdot t_{\text{obstacle}} + w_{\text{below light time}} \cdot t_{\text{below light}} + w_{\text{following line time}} \cdot t_{\text{following line}} + \\
 & + w_{\text{following below light time}} \cdot t_{\text{follow below light}} + w_{\text{total distance}} \cdot d_{\text{total}} + \\
 & + w_{\text{robot moving changed}} \cdot n_{\text{moving changed}} + w_{\text{num states}} \cdot n_{\text{states}} + w_{\text{num_trans}} \cdot n_{\text{trans}} + \\
 & + \sum_{i=1}^{\text{num scripts}} w_{\text{script_count}(i)} \cdot n_{\text{script_started}(i)} + \sum_{i=1}^{\text{num_active_areas}} w_{\text{active_area_count}(i)} \cdot n_{\text{area_started}(i)}
 \end{aligned} \quad (3)$$

Where a is an offset value, $w_{\text{attribute}}$ are weight constants of specific self-explanatory attributes. In addition, if $w_{\text{supress_score_before_load}}$ is set to 1, no scores are accumulated before cargo is first time loaded successfully. Typical values of the weight constants were established

empirically (Petrovic, 2007). In various steps, many of the constants were 0, i.e. the actual fitness function was much simpler. Figure 16 plots the best fitness average from 10 different runs. Each evaluation took into account the worst fitness for the three different starting locations and robot orientations, except of step 5, where we had to use the geometric mean of fitness from all three runs. This ensured that the behavior evolved in step 4 was not lost as the successful individuals from step 4 at least performed well when started close to the line that was leading to the loading station. Using the worst fitness resulted in losing the behavior learned in step 4 before the sensitivity to light was evolved. On the other hand, this step was repeated with the same settings, except for the use of worst fitness instead of geometric mean, before proceeding to step 6, in order to eliminate the cheating individuals from the population. In order to obtain a better evaluation of our approach, we compared the runs against an alternative scenario (which we in fact designed first, and we refer to it as sequential) of incremental steps: The robot is rewarded in different incremental steps for:

1) avoiding obstacles, 2) following a line, 3) following a line under light (while being penalized for following line outside light), 4) loading cargo, 5) loading cargo, and for following a line under light after it has loaded cargo, 6) loading and unloading cargo (one time unloading is sufficient), 7) for loading and unloading cargo (multiple deliveries are required). This sequential scenario corresponds to the sequence of skills as the robot needs them when completing the target task, being thus a kind of straight-forward sequential decomposition. Contrary to the creative scenario, here the input material in each step consists only of the individuals from the final population of the directly preceding step.

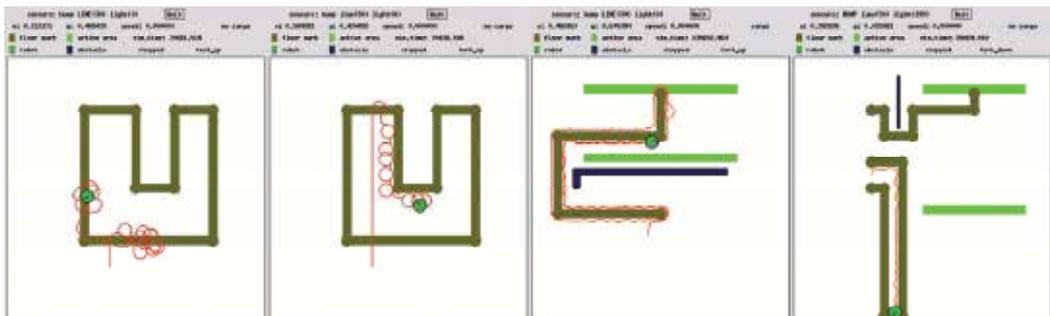


Figure 19. Examples of evolved misbehaviors demonstrating richness of controllers with FSA modules evolved for a set of predefined competence modules. In A and B, the robot is trained to follow the line. In A, when the line is encountered, it starts chaotic cyclic movements around the line, sometimes crossing a larger distance in a straight movement. In B, the robot starts following the line after it meets it for the second time, but instead of smooth following, it follows by the line by systematic looping – although actually achieving the desired behavior at certain quality level. In C and D, the robot needs to follow the line and then periodically transport cargo between the loading and unloading locations placed opposite to one another. In C, the robot loads the cargo once, but fails to stop following the line. In D, the robot fails to follow the correct (third from the bottom) line, which starts under top-mounted light (not shown in this version of viewer)

Another important difference is that the environments in all steps of sequential scenario were the same as in the final task, with the exception of the third incremental step, where the line originally leading to the loading station was changed to a loop, being illuminated by light along full its length; for this purpose we also removed one of the obstacles and

introduced an additional light source. We tried to evolve the target behavior with sequential incremental scenario without this simplification of the environment, however, even after spending several weeks of efforts and years of computational time, and exhausting the parametric space of the configuration options, and various fitness functions, the correct controller functionality was never produced. In particular, it appears to be too difficult to evolve sensitivity to light, while not losing the proper line-following behavior, if the line leaves the light and follows to the loading station, where it is non-trivial to turn and return back under the light to gain fitness. If the robots were rewarded for spending time under the light, they evolved all the possible tricks of pretending the line-following behavior, while moving in various loops, but forgetting the proper line-following behavior at the same time. Once the line-following behavior has been lost, it was very difficult for the evolution to reclaim it later again in the successor incremental steps, which required it. Modifying the environment in the third incremental step was sufficient, and the target behavior evolved in 11 of 15 runs, each run taking about 12 hours on a pool of 60 computational nodes (2 GHz PCs). Fitness plots and tables are in (Petrovic, 2007).

6. Future Work and Conclusions

We study the problem of automatic design of behavior coordination mechanism for behavior-based controllers of mobile robots by the means of evolutionary algorithms. While other researchers often investigate controller architectures inspired by neural networks, we focus on distributed coordination mechanisms based on finite-state automata. Plain evolutionary algorithms are not capable of overcoming the complexity of this design problem and therefore must be supported by additional framework. We use the method of incremental evolution, i.e. partitioning of the evolutionary task into a structure of simplified tasks of gradually increasing complexity from various viewpoints.

- We experimentally confirm that incremental evolution is a possible way of overcoming the complexity of evolutionary task in the field of Evolutionary Robotics, describe and experiment with various flavors of incrementality.
- We design and implement a new universal distributed coordination mechanism and controller architecture consisting of behavioral modules, message passing, and coordination modules associated with the behavior modules.
- We show how our coordination mechanism can be automatically designed using evolutionary algorithm with the help of incremental evolution on a non-trivial mobile robot task evolved in simulation and verified on a real robot. That means, we confirm that the behavior coordination mechanism in a behavior-based controller for a mobile robot can be automatically designed using evolutionary computation.
- In the creative scenario, various skills are learned independently in modified environments and merged together in later steps.
- In the sequential scenario, the same target environment is used, and the behavior is built in a sequential manner – from shorter sequences of actions to more complex sequences.
- The sequential scenario fails to evolve the target behavior due to a high complexity, but succeeds if the environment is modified in one of the steps.
- The creative scenario succeeds to evolve the target behavior in significantly shorter time (t-test: 2.5786).

- We show that finite-state automata, the genotype representation used in our coordination architecture, outperforms GP-tree programs on tasks with structural similarity to behavior coordination problem.
- We show that incremental evolution can both improve and decrease the evolvability; the overall effect of its use can be both positive and negative and thus the use of incremental evolution requires understanding and preliminary analysis of the problem-specific search space.

One of the important strengths of the incremental approach to evolution of controllers stems from the incremental property of the behavior-based approach to building controllers.

When a functional controller is extended with a new functionality, this is typically done by adding one or more behavior modules. The coordination of the original controller needs to be adjusted to the new task, while preserving the original functionality.

In case of distributed coordination system, this practically means modifying the previous coordination to fit the new situation, and designing the new coordination for the new module(s). It is important to realize that this is the same kind of design step as was repeated several times while designing the original controller that is being extended. Thus, our methodology for design is open-ended: the controller is never fixed-finished, rather allows for future modifications that have same characteristics, issues, and progress as the original design process.

In the future work, we would like to relax the discreteness of the state automata by combining them with probabilistic approaches, as well as to compare the approach to other methods of automatic design of BB coordination.

Simulations of mobile robotic experiments certainly form a class of hard simulation problems. The number of interactions of the simulated system (a robot) with its environment is typically extremely high, since a mobile robot must continuously scan its environment using most of its sensors. Each such sensing is a separate event, and the density of the sensor events is typically bounded only by the speed of the robot hardware – sensors and CPU. In a multithreaded system, where multiple behaviors compete for the robot CPU resources, the simulation of the robotic system becomes challenging, in the sense that even very slight differences in timing might lead to quite different robot behavior and performance. Building accurate simulating environment on a different platform is difficult and unlikely. Therefore, the controllers designed in the simulation need to be robust enough in order to cope with the transition to real robots. Extra adjustment efforts might be needed during the transition process.

6. References

- Acras, R.; Vergilio, S. (2004). Splinter: A Generic Framework for Evolving Modular Finite State Machines, *Proceedings of SBIA 2004*, pp. 356-365, Springer-Verlag
- Angeline, P.; Pollack, J. (1993). Evolutionary Module Acquisition, *Proceedings of the Second Annual Conference on Evolutionary Programming*, pp. 154-163
- Ashlock, D.; Wittrock, A.; Wen, T. (2002). Training Finite State Machines to Improve PCR Primer Design, *Proceedings of the CEC'02*, pp.13-18
- Ashlock, D.; Emrich, S.; Bryden, K.; Corns, S.; Wen, T.; Schnable P. (2004). A Comparison of Evolved Finite State Classifiers and Interpolated Markov Models for Improving PCR Primer Design, *Proceedings of the 2004 IEEE Symposium on CIBCB*, pp. 190-197

- Chellapilla, K.; Czarnecki, D. (1999). A Preliminary Investigation into Evolving Modular Finite State Machines. *Proceedings of the CEC'99, vol. 2*, pp. 1349--1356
- Clelland, C.; Newlands, D. (1994). PFSA Modelling of Behavioural Sequences by Evolutionary Programming, *Proceedings to Complex'94*, pp. 165-172, IOS Press
- Fogel, L. (1962). Autonomous Automata. *Industrial Research, 4, 2*, 14-19.
- Fogel, L. (1993). Evolving Behaviors in the Iterated Prisoners Dilemma. *Evolutionary Computation, 1, 1*, 77-97.
- Frey, C.; Leugering, G. (2001). Evolving Strategies for Global Optimization – A Finite State Machine Approach. *Proceedings of the GECCO'2001*, pp. 27-33, Morgan Kaufmann.
- Harvey, I. (1995). *The Artificial Evolution of Adaptive Behaviors*. PhD thesis, University of Sussex.
- Horihan, J.; Lu, Y. (2004). Improving FSM Evolution with Progressive Fitness Functions. *Proceedings of GLSVLSI'04*, pp. 123-126
- Hsiao, M.; (1997). *Sequential Circuit Test Generation using Genetic Techniques*. PhD thesis, University of Illinois at Urbana-Champaign.
- Jefferson, D.; Collins, R.; Cooper, C.; Dyer, M.; Flowers, M.; Korf, R.; Taylor, C.; Wang, A. (1992). Evolution as a Theme in Artificial Life: The Genesys/Tracker System. *Artificial Life II*, pp. 549-578.
- Koza, J. (1992). Evolution of Subsumption Using Genetic Programming, *Artificial Life I*, pp. 110-119.
- Lee, W.; Hallam, J.; Lund, H. (1998). Learning Complex Robot Behaviours by Evolutionary Computing with Task Decomposition. *LNCS 1545*, pp. 155.
- Lucas, S. (2003). Evolving Finite State Transducers: Some Initial Explorations, *EuroGP'03*, pp. 130-141.
- Lund, H. (2001). Co-Evolving Control and Morphology with LEGO Robots. *Proceedings of Workshop on Morpho-functional Machines*, Springer-Verlag.
- Lund, H.; Miglino, O. (1998). Evolving and Breeding Robots. *Proceedings to EvoRobot'98*.
- Nolfi, S.; Floreano, D. (2000). *Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-Organizing Machines*, MIT Press
- Maes, P. (1990). How to do the Right Thing. *Connection Science Journal, 1*, 1990, Special Issue on Hybrid Systems.
- Ostergaard, E. (2000). *Co-evolving Complex Robot Behavior*. Master thesis, Uni. of Aarhus.
- Petrovic, P. (2007). *Incremental Evolutionary Methods for Automatic Programming of Robot Controllers*, <http://urn.ub.uu.se/resolve?urn=urn:nbn:no:ntnu:diva-1748>, NTNU-trykk.
- Spears, W.; Gordon, D. (2000). Evolving Finite-State Machine Strategies for Protecting Resources, *Proceedings of the 12th International Symposium on Foundations of Intelligent Systems*, pp. 166-175, Springer-Verlag.
- Togelius, J. (2003). *Evolution of the layers in a subsumption architecture robot controller*. Master Thesis, University of Sussex at Brighton.
- Urzelai, J.; Floreano, D.; Dorigo, M.; Colombetti, M. (1998). Incremental Robot Shaping. *Connection Science, 10*, 341-360.

An Embedded Evolutionary Controller to Navigate a Population of Autonomous Robots

Eduardo do Valle Simões

*University of São Paulo – Department of Computer Systems
Brazil*

1. Introduction

This chapter studies evolutionary computation applied to the development of embedded controllers to navigate a team of six mobile robots. It describes a genetic system where the population exists in a real environment, where they exchange genetic material and reconfigure themselves as new individuals to form the next generations, providing the means of running genetic evolutions in a real physical platform. The chapter presents the techniques that could be adapted from the literature as well as the novel techniques developed to allow the design of the hardware and software necessary to embedding the distributed evolutionary system. It also describes the environment where the experiments are carried out in real time. These experiments test the influence of different parameters, such as different partner selection and reproduction strategies. This chapter proposes and implements a fully embedded distributed evolutionary system that is able to achieve collision free-navigation in a few hundreds of trials. Evolution can manipulate some morphology aspects of the robot: the configuration of the sensors and the motor speed levels. It also proposes some new strategies that can improve the performance of evolutionary systems in general.

Ever more frequently, multi-robot systems have been shown in literature as a more efficient approach to industrial applications in relation to single robot solutions. They are usually more flexible, robust and fault-tolerant solutions (Baldassarre et al., 2003). Nevertheless, they still present state-of-the-art challenges to designers that have difficulties to understand the complexity of robot-to-robot interaction and task sharing in such parallel systems (Barker & Tyrrell, 2005). Often, designers are not able to predict all the situations that the robots are going to face and the resulting solutions are not able to adapt to variations in the working environment. Therefore, new techniques for the automated synthesis of robotic embedded controllers that are able to deal with bottom-up design strategies are being investigated. In this context bioinspired strategies such as Evolutionary Computation are becoming attractive alternatives to traditional design, since it can naturally deal with decentralized distributed solutions, and are more robust to noise and the uncertainty of real world applications (Thakoor et al., 2004).

Evolutionary robotics is a promising methodology to automatically design robot control circuits (Nelson et al., 2004a). It is been applied to the design of single robot navigation circuits with some success, where it is able to achieve efficient solutions for simple tasks,

such as collision avoidance or foraging. Recently, evolutionary computation has been employed to look for solutions in multi-robot systems. In such systems, every robot can be treated as an individual that competes with the others to become the best solution to a given task (Liu et al., 2004). In doing so, the robot will have more chances to be selected to combine its parameters to produce new solutions that inherit its characteristics (i.e., spreading its genes and producing offspring, in biological terms).

Multi-robot evolutionary systems present many new challenges to robot designers, but have the advantage of a great degree of parallelism (Parker & Touzet, 2000). Therefore, the produced solutions that have to be tested one by one in a single robot system can be evaluated in parallel by every individual of the multi-robot system (Nelson et al., 2004b). In doing so, the addition of new robots to the system usually results in an increase in the performance of the evolutionary strategy, for more possibilities in the search space can be tested in parallel (Bekey, 2005).

Even though multi-robot evolutionary systems can test more solutions in the same time, the overall performance does not necessarily improve (Baldassarre et al., 2003). This is due to new factors intrinsic to multi-robot systems, such as robot-to-robot interaction. This may produce so much stochastic noise from the interactions of real physical systems that it may be impossible to the evolutionary strategy to distinguish among good solutions, which is the best one. In that context, the best solutions can suffer from the interaction with poorly trained individuals and receive lower scores, diminishing their chances to be selected to mate and spread good genes (Terrile et al., 2005).

When evolutionary systems are built in simulation, it is normally possible to exhaustively test most of the possible situations that the environment can present to an individual solution, resulting in a fitness score that better represents "how good a solution is" (Michel, 2004). With a real environment, it is very time consuming to evaluate a robot, which has to move around and react to different environment configurations. Usually, the faster the generation time, the poorest the evaluation will be. And for longer generations in the real world, the overall delay of the experiment will become prohibitive.

Additionally, the implementation of a fully embedded distributed evolutionary system often means that evolution is forced to deal with small robot populations, due to the high cost of robotic platforms (Parker, 2003). In that case, evolutionary algorithms that were designed to work in simulation with hundreds of individuals will eventually have to be redesigned to cope with these new challenges. Therefore evolutionary functions like crossover, mutation and selection will also have to be reconsidered. In such context, this work intends to present a series of experiments that investigates the effects of evolving small real robot populations, proposing novel evolutionary strategies that are able to work in such noisy environments.

2. The Implemented Evolutionary System

This session presents the strategies chosen to implement the individual controller of each robot and the evolutionary system that controls the robot team. It also shows an overview of the complete system and an introduction to the robot architecture. Although the strategies described can be applied, in theory, to control any number of robots, in this work the global idea was adapted to control a group of six robots. Even though the suggested system was proven to work with such a small population, a larger population of robots would give greater diversity to evolution, improving the performance of the system (Ficci. & Pollack,

2000). Thus, more individuals provide more genetic combinations and increase the chances of finding a good solution to the problem.

The goal of the implemented evolutionary system is to automatically train a team of six autonomous mobile robots to interact with an unforeseen environment in real time. The system is also able to continuously refine the generated solution during the whole working life of the robots, coping with modifications of the environment or in the robots (Burke et al., 2004). Although implemented into a specific group of 2-wheel differential-drive robots for a specific task, the evolutionary system can be adapted to control other kinds of robots performing different tasks. Therefore, this section is intended to be general enough to be used as guidelines to help the conversion of the system to other mobile or static platforms.

To test whether randomly initialised robots could really be trained by evolution to do something practical, a very simple task was chosen: exploration with obstacle avoidance. Such a simple task, that is also known as collision-free navigation, facilitates the implementation of the system and allows its development in relatively low-cost robots. Therefore, more robots could be built and evolution can benefit from more diversity in the population. The main issue considering functional specification in an evolutionary system is to tell evolution *what* the robots have to do, without telling it *how* they are going to achieve that (Mondada & Nolfi, 2001). In our case, the robots are encouraged to explore the environment, going as fast as possible without colliding into the obstacles or each other. Because the workspace contains various robots, the environment also includes some robot-to-robot interference (Seth, 1997) (e.g., collisions between robots and reflection of the infrared signals by approaching robots). The experiments will show how, based on a reward-punishment scheme, evolution can find unique, unexpected solutions for that problem.

2.1 The Embedded Evolutionary Controller

The robot architecture can conceptually be seen as a central control module interfacing all other functional modules, which either supply or demand data required for autonomous processing (see Figure 1). The modules were implemented using a combination of dedicated hardware and software executed by the robot microprocessor. The robot architecture is configured by a set of parameters, a certain number of bits stored in RAM memory. In evolutionary terms, this set of parameters is called the robot chromosome (Baldassarre et al., 2003). The Sensor Module is configured by a subset of the chromosome that indicates the number of sensors used and their position in the robot periphery. The motor drive module is configured by another subset of the chromosome that configures the speed levels of the robot.

The Motor Drive module receives and translates commands from the central control module and controls the direction of travel and speed of the two robot motors. The proximity of obstacles is obtained by the sensor module that decides which proximity sensors are connected to the central control module according to the parameters stored in the robot chromosome.

The Central Control Module (see Figure 1(b)) is divided into three others: the Evolutionary Control; the Supervisor Algorithm; and the Navigation Control. Connected together via the communication module, the Evolutionary Control circuits of all robots control the complete evolutionary process. They process the data stored in the chromosome and send the configuration parameters to the Navigation Control and the other modules. The

evolutionary control systems of all robots use communication to combine and form a global decentralised evolutionary system (Liu et al., 2004). This global system controls the evolution of the robot population from generation to generation. It is responsible for selecting the fittest robots (the best-adapted to interact with the environment), mating them with the others by exchanging and crossing over their chromosomes, and finally reconfiguring the robots with the resultant data (the offspring) (Tomassini, 1995).

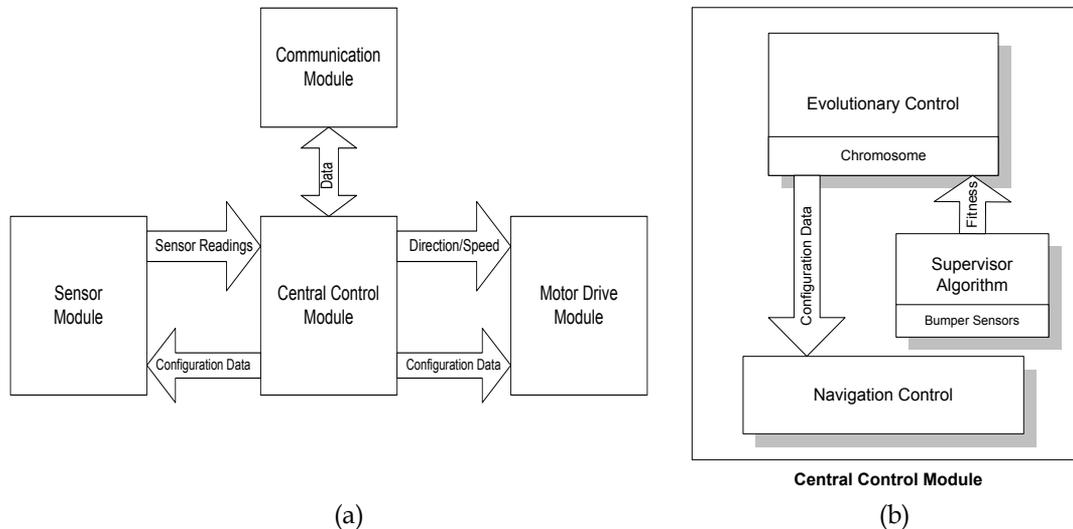


Figure 1. Architecture of the robot control system (a): the Sensor Module and Motor Drive Module are configured by the Central Control Module (b), which processes data from the sensors and commands the motor drive module in how to drive the robot

The robot performance is monitored by the Supervisor Algorithm, which informs the evolutionary control how well-adapted it is to the environment. According to events and tasks performed by the robot, perceived internally by special sensors, a score or fitness value is calculated and used by the global evolutionary system to select the best-adapted individuals to breed. The supervisor algorithm is responsible for activating a rescue routine, a built-in behaviour that is able to manoeuvre automatically the robot away from a dangerous situation once it is detected by the sensors. Contact sensors in the bumpers determine the occurrence and position of a collision. When activated, the rescue routine will take control of the robot until it is safely recovered. It can communicate directly to the motor drive module, by-passing the navigation control. When the rescue manoeuvre is completed, the supervisor algorithm allows the motor drive module to accept once more the commands of the navigation control circuit and the robot resumes on its way.

It is the Navigation Control, configured by the evolutionary control, that commands the motor drive module according to the information provided by the sensor module. It processes the information of the sensors and decides what the robot has to do. Then, it sends a command to the motor drive module, which will control the speed of the motors to make the robot manoeuvre accordingly. The navigation control is the centre of the autonomous navigation of the robot. Configured by the parameters stored in the chromosome, it drives the robot independently. Evolution is responsible for adjusting these parameters so that the robot performs well in the environment.

2.2 The Navigation Control Circuit

A RAM neural network (Ludermir et al., 1999) was chosen to implement the navigation control circuit basically because they have unique features that facilitate their evolution by the system, simplify the implementation in the robot hardware, and allow small modifications to be carried out with minimum effort. They provide a robust architecture, with good stability to mutation and crossover. Most neural networks, like the chosen one, present redundancy between the genotype and the phenotype (Shipman et al., 2000). In other words, a small change in the bits of the chromosome (the genotype) will not produce a radical change in the behaviour of the network (the phenotype). Therefore, the selected neural network is stable enough to allow evolution to gradually refine the configuration parameters of the navigation control circuit, seeking a better performance. Its good neutrality makes it suited to be evolved by the system since a small mutation on a fit individual should, on the average, produce an individual of approximately the same fitness. The RAM model does not have weighted connections between neuron nodes, and works with binary inputs and outputs. The neuron functions are stored in look-up tables that can be implemented in software or using Random Access Memories (RAMs). The learning phase consists of directly changing the neuron contents in the look-up tables, instead of adjusting the weights between nodes.

In relation to robot implementations, the RAM model, or RAM node is very attractive, since it provides great flexibility, modularity, parallel implementation, and high speed of learning, what leads to less complex architectures that can easily be implemented with simple commercial circuits. The RAM node is a random access memory addressed by its inputs. The connectivity of the neuron (N), or the number of inputs, defines the size of the memory: 2^N . The inputs are binary signals that compose the N -bit vector of the address that can access only one of the memory contents.

The RAM neural network simplicity and its implementation as elementary logic functions are responsible for its fast performance. The mapping of the RAM neural network into simple ALU logic functions and their direct execution in the microprocessor ALU can reduce even more the total memory required by the control algorithm. The faster speed provided by these simple implementations allows a faster controller, which can improve the decision rate in low-cost microprocessors.

Figure 2 shows the sensor module processing the information of the sensors and feeding the neural network inside the navigation control circuit. The output of the neural network is a command that tells the motor drive module how to control the motors. The evolutionary control reads the information contained in the chromosome and sends the parameters to configure the sensor and motor drive modules. It also reads the contents of the neurons from the chromosome and transfers them to the neural network in the navigation control circuit (Korenek & Sekanina, 2005). The motor drive module intercepts the command and activates the corresponding routine that generates the signals for the motors.

Figure 3 shows more details on how the navigation control circuit interfaces the sensor and motor drive modules. The neurons are connected in groups (discriminators) that correspond to one of the possible classes of commands (C_1, C_2, \dots, C_n) the neural network can choose. The groups are connected to an Output Adder (O_1, O_2, \dots, O_n) that counts the number of active neurons in the group. The Winner-takes-all block receives these counting from the output adders, chooses the group with more active neurons, and sends the corresponding

Command to the motor drive module. The sensor module converts the analogue readings of the infrared proximity sensors into 2-bit signals that can be connected to the neuron inputs. In the selected approach, the inputs of the RAM neurons are connected to the sensor outputs provided by the sensor module. All neurons of the network have the same number of inputs, although that number may vary according to the application (Ludermir et al., 1999). In the implemented network, all neurons in the same position in the groups are connected to the same inputs (i.e., the first neuron of the first group will have the same inputs as the first neuron of the second group and so on...).

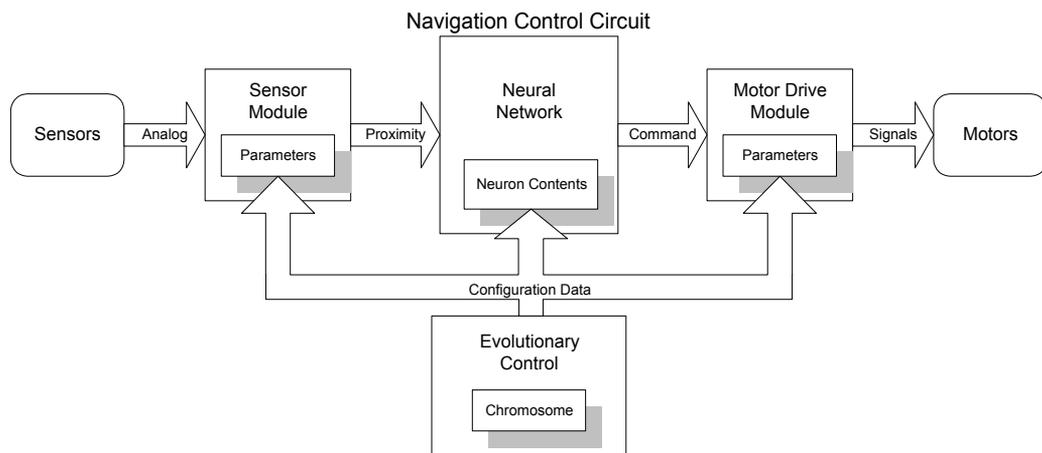


Figure 2. The navigation control circuit interfacing the sensor and motor drive modules, and the evolutionary control

The connectivity between the input lines and the sensor outputs is controlled by a Connectivity Matrix that defines which sensor outputs are connected to L_i , L_j , and L_k . The connectivity matrix is randomly initialised at the beginning of a new evolutionary experiment. One other advantage of RAM neural networks is their modularity. This characteristic simplifies the modification of the architecture. The number of neuron inputs can be modified by rearranging the connectivity to the sensors alone. Sensors can also be added or removed in this way. New commands are easily included by inserting more neuron groups.

The RAM neural network can be evolved by simply storing sequentially the neuron contents into the robot chromosome and allowing the evolutionary algorithm to manipulate these bits. Basically, the neural network must have enough inputs to cover all the sensors, although some of the sensors may be connected to more than one input line. To avoid saturation, enough neurons must be placed in the groups so that the network can learn all the different input configurations that correspond to the correct output commands. If the network is having difficulty learning a different situation, more neurons should be added. Different architectures were implemented and simulated in software until the developed solution was obtained. Figure 4 shows an example of neural network that works with four commands to control the motor drive module: *Front Fast (FF)*; *Turn Left Short1 (TLS1)*; *Turn Right Short1 (TRS1)*; and *Turn Right Short2 (TRS2)*.

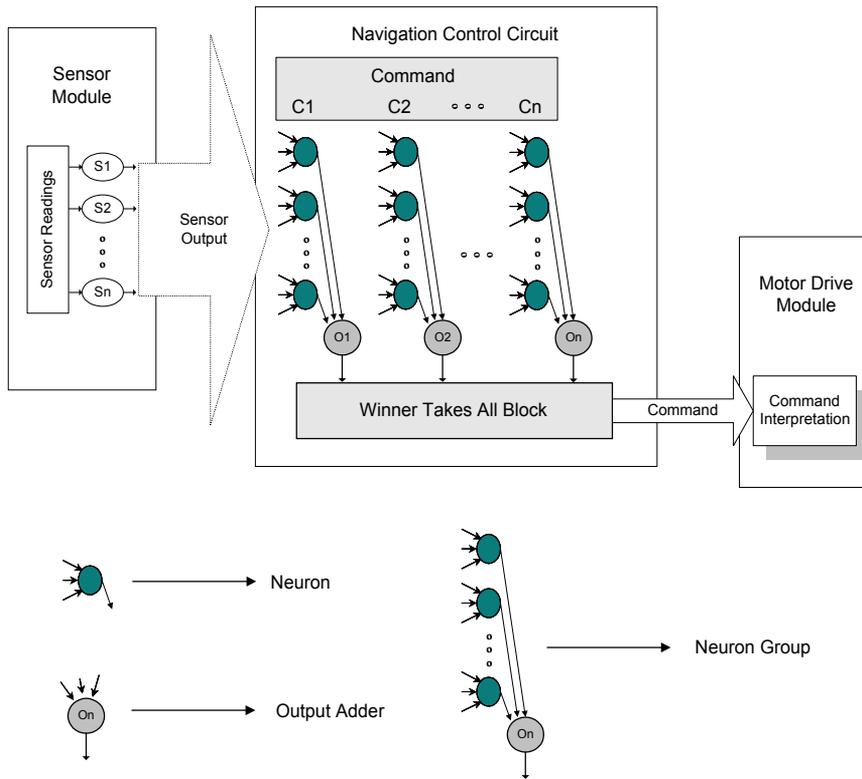


Figure 3. The neural network in the navigation control circuit. S_1 to S_n are the binary sensor readings. The Output Adders (O_1 to O_n) count the number of active neurons in the group. C_1 to C_n are the classes of commands to the motor drive module

2.3 The Evolutionary Control System

It is the evolutionary control system, located inside the central control module of the robots (see Figure 1), that performs the evolutionary processes of evaluation, selection, and reproduction (Tomassini, 1995). All robots are linked by radio, forming a decentralised evolutionary system. The evolutionary algorithm is distributed among and embedded within the robot population. Figure 5 exemplifies a cyclic evolutionary process where the individuals are evaluated according to their capacity to perform the tasks in the environment. If they perform well, it can be said that they are well-adapted to the environment. The robots are assigned a score, or fitness value, that tells how fit they are. When the evaluation period is over, the individuals select a partner to mate with according to their fitness value. The best individuals have more chance of being selected to breed. Next, they exchange their chromosomes, crossing over their genes to form the new combinations. The resultant chromosomes are then used to reconfigure the old individuals, originating new ones, or the offspring. Then, a new evaluation phase starts again. Assuming that new robots cannot really be created spontaneously, the offspring must be implemented by reconfiguring selected old individuals.

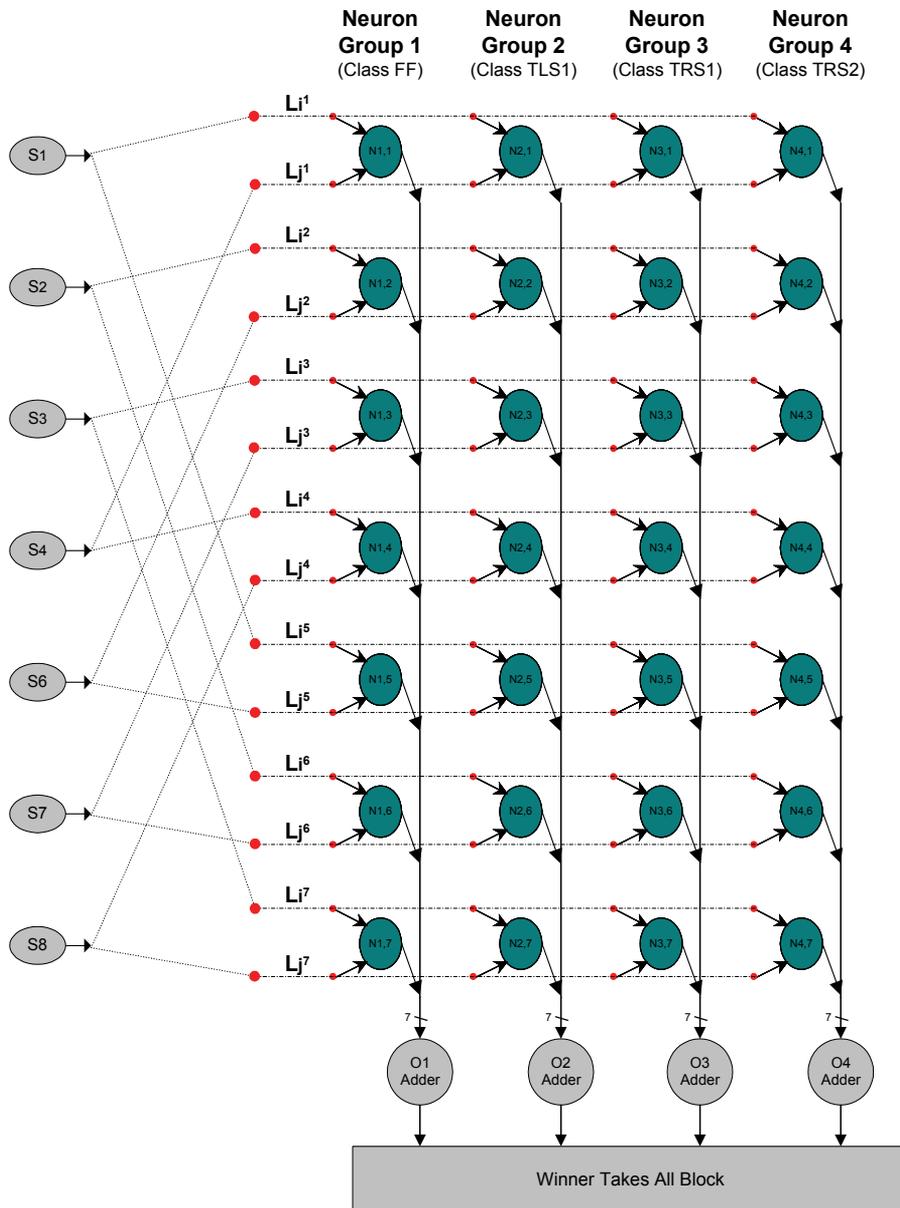


Figure 4. Configuration of the neural net with four groups of seven neurons

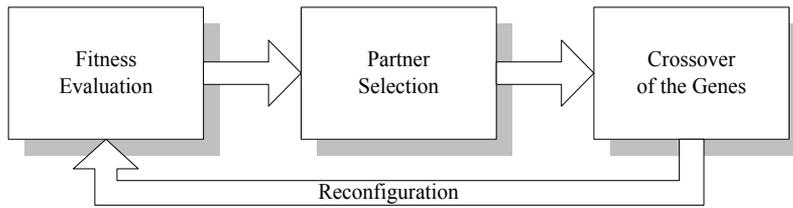


Figure 5. An evolutionary process of evaluation, selection, and reproduction (or crossover)

An evolutionary process, in the context of this work, is the procedure necessary for the development of suitable controllers for the population of robots. The process can stop when the average fitness value of the population reaches a specified threshold or continue indefinitely while the robots execute a certain task. In the developed evolutionary system, the robots work in a cyclic procedure, differently from a traditional design technique, where the controller is designed or trained at first and then transferred to the robot that is put to work. This cyclic procedure is inspired by the natural world where animals, like some birds for example, have a working or foraging season and a mating season, where they concentrate their attention in finding a mate and reproducing (Tomassini, 1995).

The cyclic procedure of the robots, a *generation* in evolutionary computation terms, is exemplified in Figure 5. The robots do not pursue reproductive activities concurrently with their task behaviour. Instead, they perform a working season, where they execute the selected task in the environment (or working domain) and are evaluated according to their performance. The internal timer of Robot 1 indicates the beginning of the mating season. It is important to observe that the evolutionary scheme is decentralised and distributed amongst all six robots. Robot 1 is by no means dominant in this process. The internal timer of Robot 1 is just used to signal the others, indicating the beginning of the mating season. It was necessary to avoid synchronisation problems, since it was impossible to guarantee that all robots would begin the mating season at the same time. In the mating season, the robots communicate to let the others know their fitness value. They start emitting a “mating call”, where they “shout” their identification, their fitness values, and chromosomes. The best robots survive to the next generation, breeding to become the “parents” of the new individuals. The less well-adapted robots recombine their chromosomes with the better-adapted ones, reconfiguring their parameters as a new robot before starting a new generation.

The robot recurring procedure for one generation, shown in Figure 5, works according to the following algorithm:

Working Season:

1. Avoid obstacles;
2. Count collisions;
3. Internal timer of Robot 1 indicates the beginning of the mating season;

Mating Season:

1. Robot 1 orders all robots to stop;
2. Robot 1 sends a mating call via the radio (containing its identification, fitness value, and chromosome);
3. Robot 2 then sends its mating call and so do all other robots, one after the other, until the last one;

4. All robots listen for mating calls, receiving every fitness value, comparing with the others, and then selecting the partner to mate with (if own fitness is the highest, the robot does not breed);
 5. When all genes are received and partners chosen, start Crossover;
 6. Begin reconfiguration with the resultant chromosome and wait until...
7. Robot 1 announces the end of the mating season and orders all robots to start another cycle.

The process begins with the random initialisation of the robot chromosomes. Then, the first generation starts with all robots performing their tasks in a working season. For the case of obstacle avoidance, they will navigate and have their fitness value calculated according to a function similar to the one presented in Figure 6. Robot 1 has control over the duration of the working season and uses the radio to stop the other robots when its internal timer reaches the end of the working season or the "lifetime" of the robots. That is important to synchronise the cycle and make sure that all robots will stop working at the same time. Starting with Robot 1, each robot transmits, one by one, a mating call via the radio, containing its identification, fitness value, and chromosome. When they are not transmitting, the robots listen for other mating calls, receiving the fitness value from the call, comparing it with the others, and then selecting the optimal partner with which to mate. If own fitness is the highest, the robot does not breed and "survives" to the next generation. The cycle will be completed when all robots find a partner to mate with and combine their genes in the crossover phase. The mating season lasts until all the six robots signal Robot 1 that they have found a partner, have mated, and have reconfigured themselves with the resultant chromosome. Robot 1 then orders them to restart another cycle (once more Robot 1 is used only to synchronise the next phase). In other words, the best-adapted robots "survive" to the next generation, while the others "die" after mating, to lend their bodies to their offspring.

2.4 Fitness Evaluation

A simple obstacle avoidance task was chosen. The limited complexity of this task allows a good evaluation of the fitness in a short period. A complex task requires more time so that the robots can be subjected to more challenges in order to show that they can perform well in more than specific situations. A smaller generation time means a faster evolution, because more combinations of solutions will be tried (Pollack et al., 2000).

A reward-punishment scheme is applied during the fitness evaluation process, executed by the supervisor algorithm. Each robot is evaluated during the "*working season*", where its fitness function is calculated by penalising collisions and lack of movement (reducing the fitness value), encouraging the exploration of the environment (rewarding by increasing the fitness value for every second of movement). A major issue that must be addressed is how to detect a good (fit) robot. This question may be highly complex in nature, but in the context of evolutionary systems, it can be simply defined by the programmer, in accordance with the particular problem at hand. Furthermore, writing a fitness function depends on the targeted behaviour and the characteristics of the robot, and the necessary insights are gained through incremental augmentation over many trials in the environment.

For the obstacle-avoidance problem, a simple rule can be applied: a robot will increase its fitness each time it comes across an obstacle and successfully avoids it. Each time it collides, the fitness will be decreased. Figure 6 shows an example of a fitness function where the

robot fitness is increased by one for every second the robot is in movement, encouraging exploration. It is punished by decreasing its fitness by ten when it collides. The fitness is also decreased by 100 to punish the robot for turning for more than five seconds. Therefore, this sub-function prevents a particular efficient solution that kept the robot spinning in a small circle within an obstacle-free area.

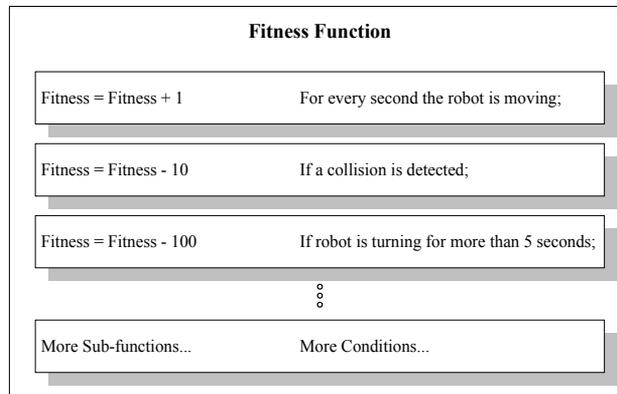


Figure 6. An example of how a fitness function can be constructed

In a situation where an obstacle is close to the robot, but the proximity sensor readings are not interpreted correctly by the navigation control or are not enabled by the sensor module, a collision may occur and the fitness variable will be decreased. The bumper sensors will then be analysed by the supervisor algorithm to calculate where the collision took place in a total of 12 sectors with 30 degrees each. Once the place of collision is detected, a rescue routine will drive the robot away from the obstacle, returning the navigation control to the neural network. When the robot is moving forward without colliding with obstacles, its fitness will be increased every second.

2.5 Partner Selection

The procedure for partner selection is based on the robot fitness value or score. They can be very simple, like choosing the best robot to mate with all other ones, or more complex, such as the roulette-wheel technique (Tomassini, 1995). In this work, the simple approaches are preferred because of the restrictions of an embedded controller (i.e., low processor speed and small memory size). As the population is very small, a simple technique can deal with the robot selection without problems. Therefore, some simple, but efficient selection techniques were developed. These techniques are:

1. Select the robot with the highest fitness value in the generation to breed with all other robots and survive to the next generation. This tries to make sure that in the next generation the best fitness will be at least similar to the present one.
2. An "Inheritance" scheme was developed: the score used to select the robot is the average of the robot fitness in the last five generations (i.e., inheriting the scores of its previous generations). The robot with the best average survives, but only breeds with the robots with the fitness in the present generation lower than its own fitness. This approach protects new robots that are actually better than the one with the highest average, but need more generations to be selected by their average.

3. Another very simple strategy that was effective in the experiments is to select the fittest robot, allow it to survive, and reconfigure all the others with a small variation (mutation) of its chromosome. This is a form of “asexual reproduction”, where the robots do not cross over their chromosomes. All robots in the next generation will be a copy of the best one, but will suffer random changes (mutation) in a few genes.

All techniques suggested above are *elitist*. Elitism requires that the current fittest member (or members) of the population is never deleted and survives to the next generation (Tomassini, 1995). The developed inheritance scheme prevents a robot from being deleted even if it is not the fittest, but has the biggest accumulated average fitness value. It is the only selection technique where the fittest member of the population does not have the same number of offspring (or the same probability to have offspring) whether it is far better than the rest, or only slightly better. In the other ones, it will always have the same probability to have offspring; and in most of them, will breed with all other robots. This approach is often too severe in restricting exploration by the less fit robots.

A common problem with these techniques is the possible appearance of a super fit individual that can get many copies and rapidly come to dominate the population, causing premature convergence to a local optima (Mondada & Nolfi, 2001). This can be avoided by suitably scaling the evaluation function, or by choosing a selection method that does not allocate trials proportionally to fitness, such as tournament selection. This work, however, does not experiment with these methods.

2.6 Reproduction Strategy

The crossover is the phase in the evolutionary algorithm where the chromosomes of both parents are combined to produce the offspring (Tomassini, 1995). Many techniques are proposed in the literature to implement the crossover phase. Nevertheless, this work uses a very simple strategy, because of the restricted resources of the embedded controller.

In the developed evolutionary system, both morphological features and the controller circuit are evolved to respond to changes in the environment. The robots constantly adapt to changes in the surroundings by modifying their features and the contents of the RAM neural controller. The term “morphology” is defined as the physical, embodied characteristics of the robot, such as its mechanics and sensor organisation. In the performed experiments, the morphological features modified by evolution are the number and position of sensors, as well as the speed levels of the drive motors. Therefore, the genetic material specifies the configuration of the robot control device and morphological features. Eight pairs of genes in the chromosome ($B1$, $B2$ to $B15$, $B16$) are used to configure the sensor module; ten genes ($B17$ to $B26$) configure the motor drive module; and the remaining genes ($B27$ to Bn) configure the navigation control module (neuron size \times number of neurons for a RAM neural network). The control device is implemented within the robot microprocessor (a neural network for navigation control) and two programmable modules control the robot features, which are the sensor module and the motor drive module (refer to Figure 1).

For the selection of the robot features controlled by the sensor module, a more complex “dominance approach” was implemented to combine the eight pair of genes. Each sensor in the sensor module is configured by two genes in the chromosome: *i*) two genes will determine the presence of a feature (“enable the sensor”); *ii*) one gene comes from each parent; and *iii*) all features are recessive. The two genes are coded using bits in such a way that the combinations “1,1”, “0,1”, and “1,0” disable the sensor, and “0,0” enables it.

The motor drive module has ten bits associated with it in the chromosome. The features selected by the module are the three different speed levels for the motors. Figure 7 shows that the three speed levels, *Fast*, *Medium*, and *Slow* are controlled by these ten bits. The contents of these ten bits (B_{17} , B_{18} , B_{19} , B_{20} , B_{21} , B_{22} , B_{23} , B_{24} , B_{25} , and B_{26}) are added together to form a decimal number that indicates the speed level *Fast*. The result is a number between zero and ten that indicates the level of the *Fast* speed. The contents of the first six bits (B_{17} , B_{18} , B_{19} , B_{20} , B_{21} , and B_{22}) are added together to form a decimal number that indicates the speed level *Medium*. The result is a number between zero and six. In the same way, the contents of the first three bits (B_{17} , B_{18} , and B_{19}) are added together to form the decimal number that indicates the speed level *Slow*. The result is a number between zero and three. Therefore, as they use the same bits, this guarantees that the level *Fast* is always greater than or equal to *Medium*, which is always greater than or equal to *Slow*.

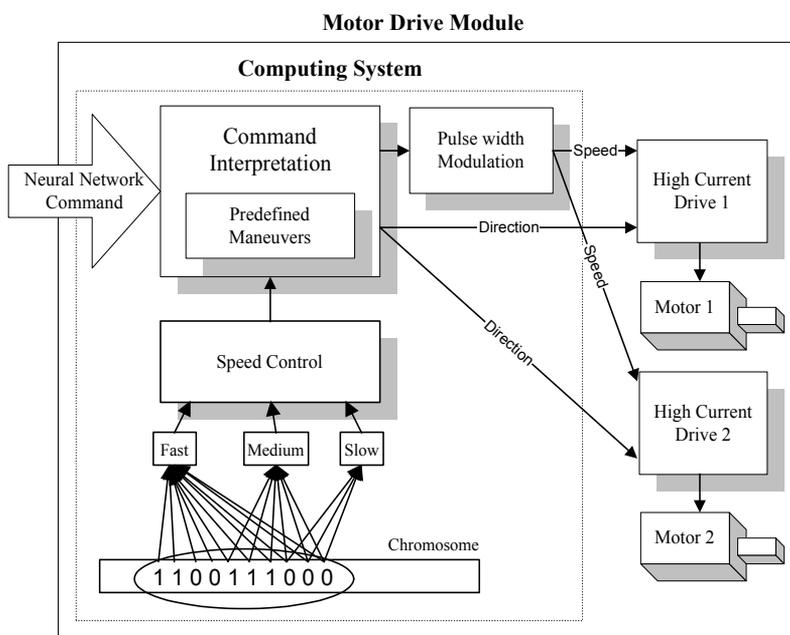


Figure 7. The motor drive module. The speed levels *Fast*, *Medium*, and *Slow* are specified by the chromosome. The command interpretation block has a set of predefined routines to drive both motors. The microprocessor implements most of this module, including the pulse-width modulation signals that control the motors

The resultant speed for the three levels is converted by the motor drive module to a value (an internal parameter) between 1 and 32, because the robot motor can have 32 speed levels. This conversion is not linear, because of the way the motors are controlled by pulse modulation, and the corresponding values are shown in Table 1. This approach permits co-adaptation where the chromosome integrates specifications for both controller and morphological features. Evolution can select not only the number of sensors to use, but, if the number of sensors is fixed, it can select which ones to pick (i.e., the sensor position on the robot).

Speed Level:	0	1	2	3	4	5	6	7	8	9	10
Internal parameter:	1	7	9	11	14	17	20	23	26	29	32
Velocity (m/s):	0	0.02	0.05	0.08	0.1	0.13	0.15	0.17	0.2	0.23	0.26

Table 1. Conversion of the speed levels of the robot

For the genes that control the neural network and the motor drive module, a random exchange of the genes from the parents is used to form the resultant chromosome. This strategy is called *uniform crossover* (Tomassini, 1995), although here only one offspring is produced. Therefore, a gene is selected from the father or the mother to occupy the corresponding position in the offspring chromosome. Thus, a random exchange of genes from both parents occurs after a dominance selection of the 16 bits that enable the sensors.

After the crossover is completed, a mutation phase starts. Applying mutation to a chromosome means that a small number of copying errors may occur when copying the genes from the parent chromosomes to the offspring. In this work, a mutation rate of $M\%$ means that each gene in the chromosome has a probability of $M\%$ of being selected and binary inverted (e.g., new gene = $NOT(\text{gene})$). Small mutation rates, usually between 1 and 3%, are the ones that produced the best results in the experiments. Higher mutation is only useful in the beginning of the evolutionary experiment. A high mutation rate does not help to evolve faster, and did not prove to be a good strategy.

3. System Overview

The mobile autonomous robots form a decentralised embedded evolutionary system where no host computer is required. Nevertheless, an IBM PC is used to monitor all data exchanged via the radio link, producing a complete record of the chromosomes, parameters, and variables for every generation. The robots can communicate with each other and the monitor computer via an asynchronous serial data link using their communication module. The computer monitors the robot internal variables without interfering in the system, but has the capacity to start or stop an evolutionary experiment.



Figure 8. The monitor computer connected to the radio

A radio board is connected to the monitor computer, which logs data on the evolutionary experiment. It is a multi-channel driver/receiver interfacing the IBM PC via its serial port. A software interface permits the downloading of software, data, and commands between the robots and the computer. Programming and low-level debugging are provided by the computer. When programming or debugging, bi-directional data between the robot

processor and the computer can operate either via a wired link or via radio. Figure 8 shows the monitor computer connected to the radio board that is used to communicate with the robots and monitor the evolutionary experiment.

The experiments were performed within a $2.50\text{m} \times 2.50\text{m}$ working domain, containing walls and obstacles of varied sizes, where the robots can explore the environment, avoiding collisions. Many movable obstacles and internal walls of different sizes are available to change the scope of the workspace where the robots navigate. The workspace can be modified into different configurations by rearranging the obstacles and walls. This flexibility is necessary to allow different degrees of complexity of the environment during the experiments. Figure 9 shows different configurations of the workspace, representing a simple (a) and a complex (b) environment.



Figure 9. An example on how simple (a) and complex (b) environments can be produced by rearranging the obstacles and walls

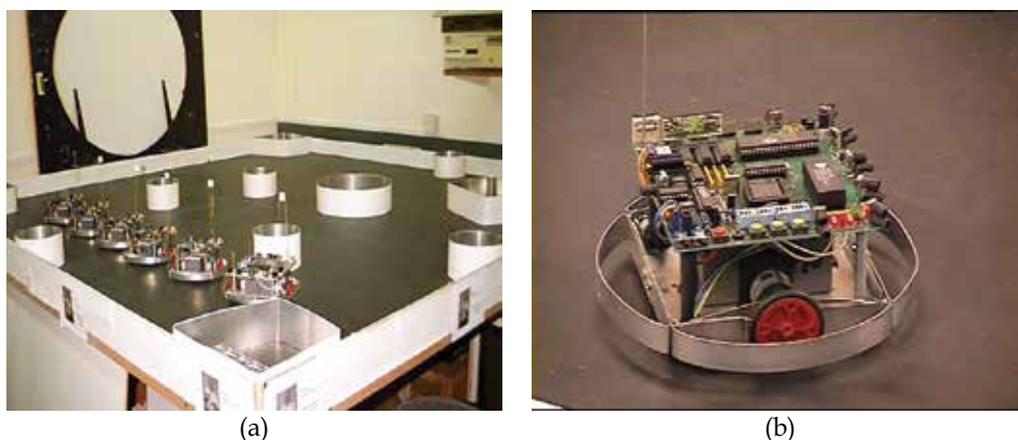


Figure 10. A view of the robot team (a) and top-view of Robot 2 (b), showing the position of the infrared sensors and bumpers

The robot architecture consists of a two-wheel differential-drive platform (20cm diameter), containing a Motorola 68HC11 - 2MHz, 64Kb of RAM, bumpers with eight collision sensors, and eight peripheral active infrared proximity sensors. It exchanges information with the other robots at 1.2Kbps by a 418MHz AM radio. Both robots and workspace were specially built for the experiments. All eight proximity sensors are connected to the sensor module, which is configured by the chromosome. The module can individually enable or disable the

sensors, changing the number of active sensors and, consequently, their position in each generation. Therefore, the physical, embodied characteristics of the robot can be modified. The wheels are placed in the middle of the robot, allowing it to turn around its central axis. The robot has four round bumpers on its front, back, left, and right, which are attached to the base by eight contact sensors that permit the supervisor algorithm to pinpoint the location of a collision.

Figure 10 (a) shows the team of robots parading in their workspace. Figure 10 (b) shows a top-view of Robot 2, displaying its infrared proximity sensors and round bumpers. The three green keys on the right form a small keyboard used to enter commands to the robots manually.

4. The Experiments

4.1 Experiment 1: Evolving with a Simple Fitness Function

In this experiment, the embedded evolutionary system attempts to evolve the control circuits of the group of six robots. The eight sensors were permanently enabled and the speed of the motors was fixed at maximum speed. Therefore, the navigation control circuit was the only one under evolutionary control. All sensors were set to operate at *medium* range, detecting obstacles closer than 15cm. The sensor positions around the robot are presented in Figure 11.

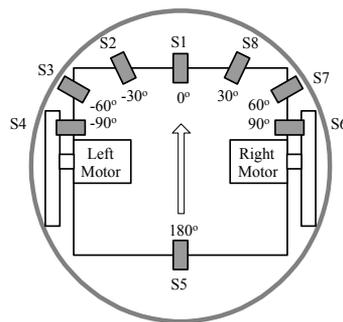


Figure 11. Position of the sensors on the robot and their angle in relation to the central line of the robot

The neural network can take seven commands to control the motor drive module: *Front Fast* (FF); *Turn Left Short1* (TLS1); *Turn Right Short1* (TRS1); *Turn Right Short2* (TRS2); *Turn Left Short2* (TLS2); *Turns Right Long* (TRL); and *Turn Left Long* (TLL). *Front Fast* means “move forward with maximum speed”. To turn left/right short, the robot moves with reverse direction in one of its motors (with both motors at maximum speed), causing a spin around its own axis. The difference between TRS1 and TRS2 is that in the later, the robot keeps turning for 200ms, while the duration of the other three commands is just one iteration (10ms more or less). In TRL and TLL, the robot turns right/left by breaking one wheel and turning around it with the other one in maximum speed in a wide arch of one wheel span of diameter. The speed of both motors was fixed at the maximum setting, enabling the robots the move at 0.26m/s. As the selection of the sensors and the speed levels were fixed, the only feature that varied was the controller configuration.

Using only seven output commands means that the neural network needs only three bits to encode them and seven classes of discriminators to work. The neural network architecture shown in Figure 4 is preserved and three new groups of seven neurons were added too represent the new commands. Therefore, the architecture has now seven groups of 7 neurons ($m=7$ and $n=7$) with two inputs each (the neuron size is four bits). The interconnections between sensors and neurons are randomly chosen. Each neuron has four bits of memory to store its contents. The neural network has 49 neurons with a total memory size of 196 bits. This would also be the size of the corresponding bit string in the chromosome, if the neural network was to be evolved. The winner-takes-all block chooses the command that has more active neurons and encodes it with three bits, before sending it to the motor drive module. Therefore, if the size of the genotype of the robots is 196 bits, then the current search space is considerably large: $2^{196} = 1.004 \times 10^{59}$.

Aim of the Experiment:

This experiment aimed to test if the embedded evolutionary controller was able to evolve the navigation control circuit until the collision-free navigation behaviour emerges, while having the robot morphology (the sensor configuration and motor speeds) fixed. To achieve this, the sensor configuration was fixed with all sensors enabled and the velocity of both motors was fixed at maximum speed. The navigation control circuit is configured by only 196 bytes in the robot RAM memory and evolution is allowed to manipulate any bit of these bytes. The robots were allowed to reproduce and suffer mutation, but only with the bits in the chromosome that correspond to the navigation control circuit.

Experimental Setting:

The population of robots with different configurations was evaluated in 60-second generations. The robots were allowed to reproduce and mutate with mutation rate equal to 3%. The chosen selection strategy for this evolutionary experiment was:

- *Select the robot with the highest fitness value in the generation to breed with all other robots and survive to the next generation.*

This strategy means that the robot with the highest fitness is selected, and will send its chromosome to the other five robots. Each one of the remaining five robots then combines its own chromosome with the one received from the best robot to produce a resultant chromosome. Then, the remaining five robots reconfigure themselves with the mutated chromosomes and the robots continue in the next generation.

In the crossover phase, where the two chromosomes are combined to form a resultant offspring, each bit is randomly chosen from the corresponding location in the chromosome of the parents. Then, in the mutation phase, a random number r is generated between zero and 100 for each bit in the chromosome, and the bit will be flipped each time r is smaller than the mutation rate.

This experiment used a very simple fitness function in an attempt to prevent biasing evolution to any preconceived idea of an ideal controller. The selected fitness function for this test was:

1- Start with 4096 points;

2- Reward: increase fitness by 10 points every 1 second without collision;

3- Punishment: decrease fitness by 30 points for every time command is not FF for more than 15 seconds;

4- Punishment: decrease fitness by 10 points for every collision if command = FF, TLL, or TRL.

Rule 2 constantly rewards the robot with five points for every second it is moving without colliding. Rule 3 was introduced to prevent evolution from producing a solution that keeps

turning around itself and never collided. This rule punishes the robots that keep turning for more than 15 seconds, encouraging them to move forward. Rule 4 only punishes the robots that are not turning around themselves, since a collision in that case can only mean that they are being crashed by another robot. If a robot is executing any command but *FF*, *TLL*, or *TRL* its centre is not moving, so the collision cannot be its fault.

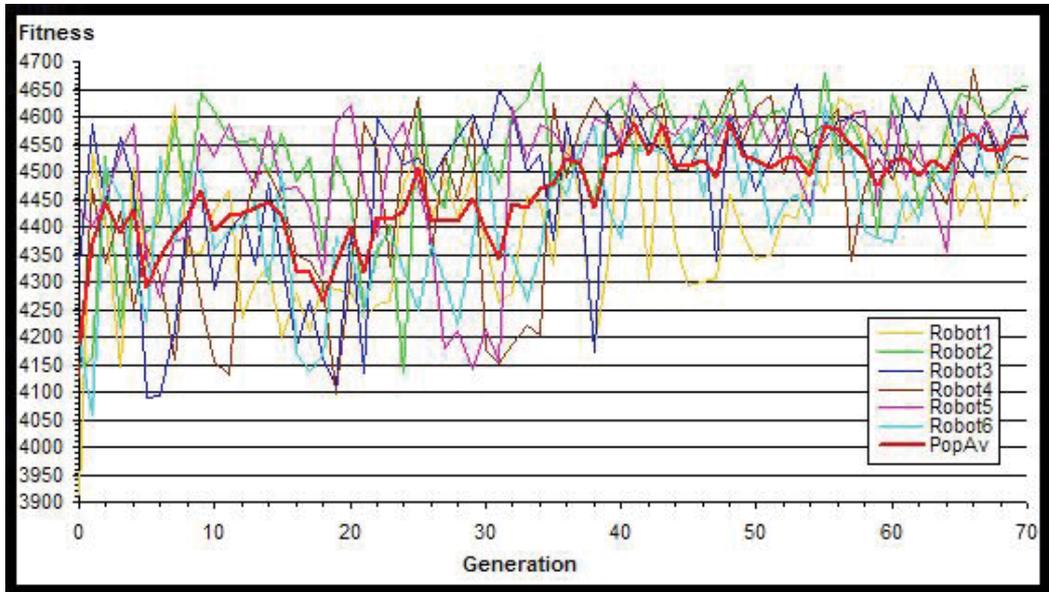


Figure 12. Evolution of the *neural network controller* using mutation rate of 3%, and generation time of 60s. The population was randomly initialised. *Robot n* is the fitness of Robot n and *PopAv* is the average fitness of all robots in the generation

The experiment in Figure 13 was performed to investigate if the robot population, once initialised with a well-trained controller, could hold this configuration or would degenerate. The robots were initialised with a previously trained neural network, which was hand-designed to avoid obstacles in every situation the robots can face. The population was then allowed to mate and mutate with the same parameters of the previous evolutionary experiment to determine if it would ever get to such a good solution if the experiment was allowed to continue for more generations. The results presented in this figure can be compared to the ones obtained from the evolution of a randomly initialised population in Figure 12.

Discussion:

Figure 12 showed that the evolutionary system succeeded in evolving the population towards the expected behaviour, producing solutions that practically did not collide after 35 generations. It was observed in this experiment that one could not rely only on the fitness of a robot to know how well-adapted it is. A simple solution can be lucky enough to start its lifetime in a safe part of the environment, away from obstacles and other robots, which would produce a very high performance. This was observed in some robots during the evolutionary experiment (e.g., Robot 2 that scored 4644 in generation 10, or Robot 1 that scored 4621 in generation 8). Therefore, human judgement was the best way to evaluate the abilities of the robots and the performance of the evolutionary system.

In the experiment shown in Figure 12, all sensors were enabled from the beginning of the evolutionary experiment. The controller had information from all the sensors and needed to learn what to do with it. Based on observation, some robots learned how to use the frontal sensor, *S1*, which gave them an advantage early on in the process. This was the case for Robot 2 that could avoid obstacles detected by *S1* from generation 8. Robot 2 quickly became the best robot and throughout the crossover operation transferred to Robot 5 the ability of using *S1*. Robots 2 and 5 dominated the population until generation 23, where they were overtaken by Robot 3 and Robot 4. Robot 3 learned how to use *S1*, *S4*, and *S7* from generation 29, and was able to avoid most of the obstacles very well thereafter. From generation 47 on, all robots acquired the necessary skills to employ at least three sensors and the average performance was much better. From generation 50, Robot 2 was well-adapted to the environment, and could use *S1*, *S2*, *S4*, *S6*, and *S7* to avoid collisions with most of the obstacles and other robots.

The major observed problem was the instability of the system, where there was no guarantee that a good solution, such as Robot 2 in generation 50, would be selected as the best robot until it is outperformed by better robots. The population performance dropped when a robot with a poorly-adapted controller (Robot 1) was lucky enough to be selected as the best robot and spread its bad genes through the population. Robot 1 had a poorly-adapted controller that made it move forward each time *S6* detected an obstacle, independently of having something in its way. The average performance dropped, but after a few generations, the fitness of the population recovered.

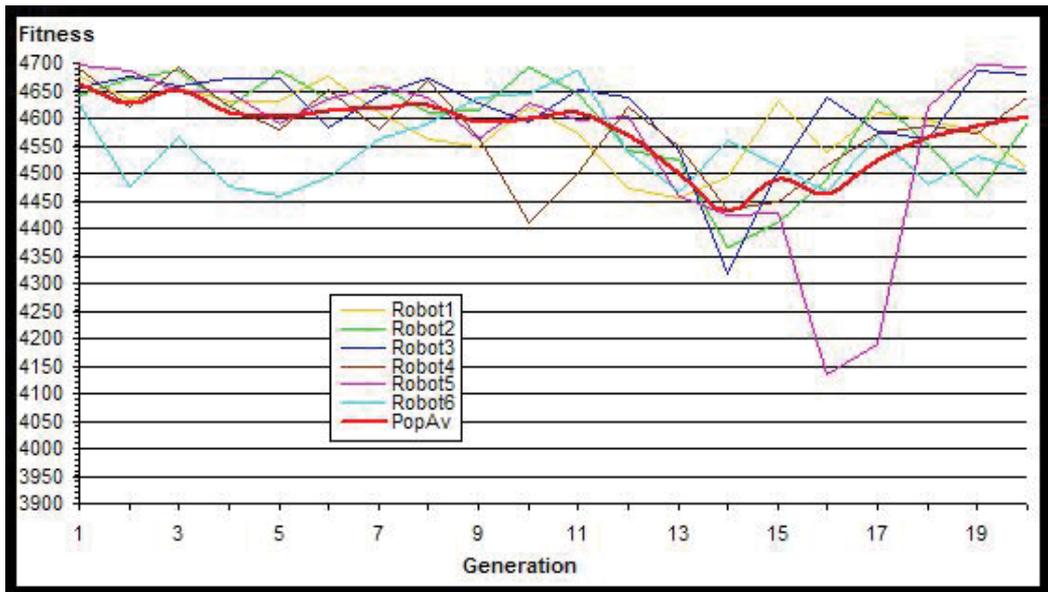


Figure 13. Evolution of the *neural network controller* using *mutation rate* of 3%, and *generation time* of 60s. The population was initialised as a hand-designed controller. *Robot_n* is the fitness of Robot *n* and *PopAv* is the average fitness of all robots in the generation

The average of the population fitness is a good parameter to determine if the system is converging to the desired behaviour. The population performance oscillated, but kept improving through the evolutionary experiment. With such a large search space (1.004×10^{59}), a perfect robot that can deal with all sensors should take a very long time to

obtain with this evolutionary approach. Nevertheless, the system succeeded in producing an even population of robots that can successfully avoid the obstacles in some of the situations faced and produce a fitness near to the maximum performance (4696).

Figure 13 shows how the system behaves after the population has been initialised with a hand-designed controller. The aim of this test was to show that if such a good solution happen to be produced by evolution, it would be preserved in the process. If the evolutionary experiment cannot keep such a solution, it is unlikely that it will ever be produced by evolution under these circumstances. Unfortunately, the figure showed that the evolutionary system, as set in this experiment, could not keep a good solution for more than ten generations and the performance of the system degenerated. Although it was possible to recover and continue to improve performance after the loss of the hand-designed controller genes, this test indicated that it is unlikely that such a good solution will be produced by the process as it was configured for this experiment. Maybe a more biasing fitness function can produce a better result.

4.2 Experiment 2: Evolving with Inheritance

This experiment developed a different selection strategy for evolutionary systems that was defined as *Inheritance Strategy*. From now on, the robots will be selected to breed based on not only their performance in the current generation, but also on their fitness in the previous generations. Therefore, the robots *inherit* part of the points scored by their predecessors. By considering the average fitness scored by the robot in the previous generations, this strategy attempts to reduce the effect of chance, which can produce bad performances if the robot is “unlucky”. This experiment tested this new approach, and evaluated if it could solve the instability problem pointed out by Experiment 1.

Aim of the Experiment:

The aim of this experiment is to develop and test a novel selection strategy involving inherited scores. It evaluated whether this new strategy could solve the problems with instability presented by the previous selection strategy, where a lucky unfit robot can be selected as the best robot instead of better ones that by chance started in a more crowded area of the environment.

Experimental Setting:

Apart from a different selection strategy, this experiment was set exactly as Experiment 1. The chosen selection strategy for this evolutionary experiment was:

- *The score used to select the robot is the average of the robot fitness in the last three generations (i.e., inheriting the scores of its previous two generations). The robot with the best average survives, but only breeds with the robots with the fitness in the present generation lower than its own fitness.*

This approach favours the robots with the highest average, which are the ones that have performed well for the last three generations. If in the current generation one lucky unfit robot happens to achieve the highest score, it is still unlikely that it will have the best average score and will not be selected to mate. This was an attempt to improve the stability of the system and reduce the effect of noise and interactions among robots and obstacles.

The charts display the fitness of the six robots in the current generation as well as the average fitness value ($AvRn$) scored by each robot in the current and the previous two generations. For each robot:

$$AvRn = (FitnessRn_{G0} + FitnessRn_{G-1} + FitnessRn_{G-2})/3$$

In the equation above, n is the number of the robot; $FitnessRn_{G_0}$ is the fitness of the Robot n in the current generation; $FitnessRn_{G-1}$ is the fitness scored by Robot n in the previous generation; and $FitnessRn_{G-2}$ is the fitness scored by Robot n two generations before.

Results:

This is a strategy that protects a fit robot in the current generation that scored more than the best robot (the one with the highest average) by not allowing it to mate. If the robot with the best average is allowed to breed with robots with fitness in the present generation lower than its own fitness, novel better solutions will not be selected because their average will be smaller than the one of the robot that won for the last three rounds. This can destroy the precious good genes of this robot and evolution will lose this better performance.

This approach protects new robots that are actually better than the one with the highest average, but need to be evaluated for more generations to be selected by their average. If one of these protected robots has a better configuration, it will probably repeat its good performance, and within one or two generations its average fitness may be the highest and the robot will be chosen to reproduce. If the protected robot is in fact a lucky unfit one, it may not repeat its good performance and, if so, will be allowed to breed in the next generation. Figure 14 shows a test with the updated selection strategy.

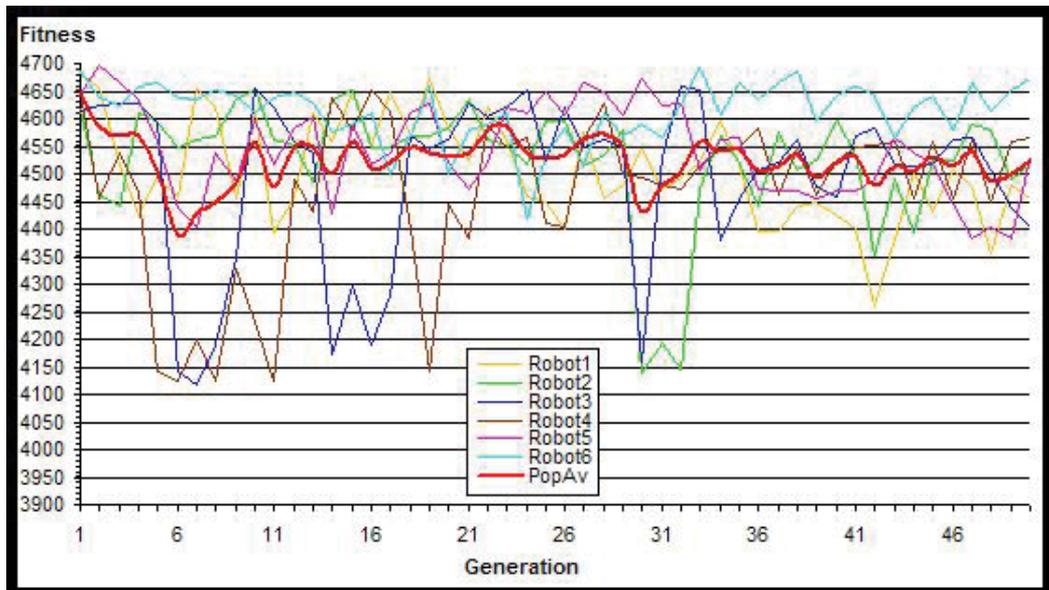


Figure 14. Evolution of the *neural network controller* using inheritance selection, mutation rate of 3%, and generation time of 60s. The population was initialised as a hand-designed controller. $Robot_n$ is the fitness of Robot n and $PopAv$ is the average fitness of all robots in the generation

Discussion:

The new selection strategy presented in Figure 14 succeeded in improving system stability. The fitness of the population dispersed mostly because of mutation, but the performance of the best robot did not degrade as much as in the previous attempts. It now protects the potentially better configurations, as happened with some of the robots that outperformed the robot with the best average and were given the chance to repeat their better

performance. When they managed to win again, their average became finally higher and they were selected as the best robot of the generation, breeding with all other robots. In the next experiment, presented in Figure 15, this new selection strategy is applied to evolve a randomly initialised population.

Figure 15 shows that even now the evolutionary system could not prevent the population performance from degrading due to mutation. This figure showed that in 50 generations, the population performance degraded to a level similar to the one where the experiment of Figure 14 stopped, showing that it is unlikely that it will improve the population much more than this level.

This experiment showed that the new inheritance selection helped to improve system performance, but did not completely solve the instability problem. The system does not seem to be able to produce an optimal solution to the collision-free navigation task. The experiments demonstrated that evolution still can take a considerable amount of time to achieve a solution as good as the hand-designed controller and may not even be able to get there.

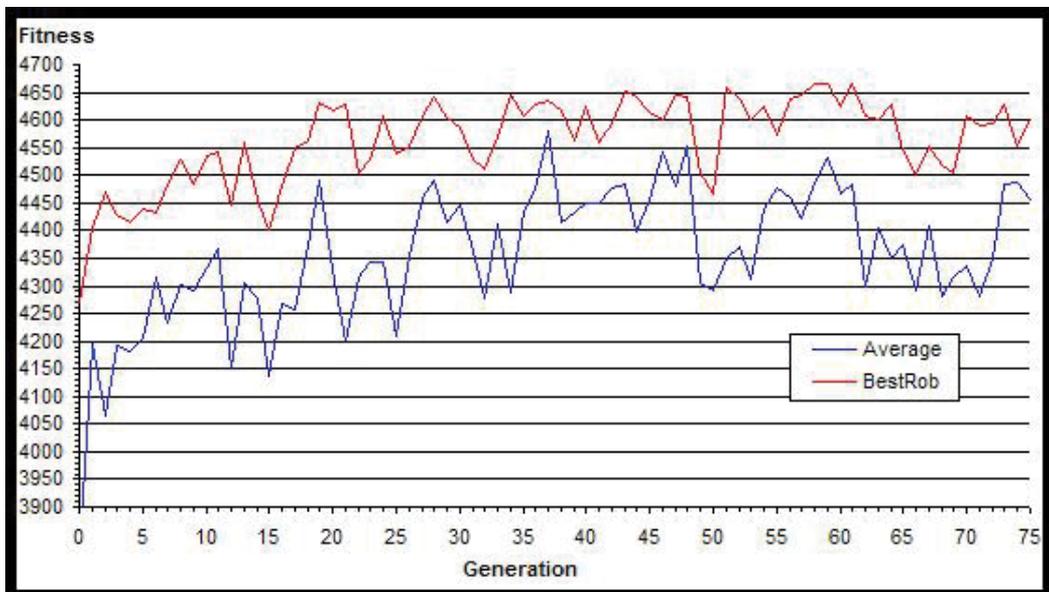


Figure 15. Evolution of the *neural network controller* using inheritance selection, mutation rate of 3%, and generation time of 60s. The population was randomly initialised. *Average* is the average fitness of all robots and *BestRob* is the fitness of the best robot in the generation

4.3 Experiment 3: Disabling Back Mutation

In the previous experiments, mutation was randomly choosing which bits in the chromosome to change. This meant that any bit in the chromosome could mutate to a different value and later mutate back to its original value. In the scope of this work, these events were called *back mutations* (Tomassini, 1995). This fact was causing evolution to waste time trying to find a good solution by testing some configurations that have been tested before. Consider a chromosome that has 99% of its bits set to the correct ones. If any bit can be changed by mutation, the chance of changing the bad ones is only 1%. This means that

99% of the time, this strategy produces configurations that are potentially worse than the current chromosome, and many generations are wasted in evaluating them.

To prevent evolution from wasting time in evaluating configurations that have been tested before, a new strategy that prevents back mutations is developed in this experiment. It consists of marking each bit in the chromosome that suffered mutation and only allowing the bits that are not marked to mutate. To achieve this, a binary array was created in memory with the same size of the chromosome. It is initialised with zeros and when one bit in the chromosome is mutated, "1" is written to the corresponding position in the array. Only the bits in the chromosome that have zeros in the corresponding position in the array are allowed to mutate. Once all bits have mutated, the array will be full of "1s". Then, the evolutionary system resets the array to "0s" and every bit in the chromosome will be allowed to mutate again. This strategy was called *back-mutation prevention*. By preventing back mutations, this strategy forces evolution to evaluate the effect produced by each bit in the chromosome.

Aim of the Experiment:

The aim of this experiment is to apply a new strategy that prevents back mutation of the bits in the chromosome. It tested whether this strategy could be applied in sexual reproduction and provided data to analyse which is the best solution for evolving the robots in the long term.

Experimental Setting:

This experiment used the same settings of Experiment 2 where the evolutionary system is working with inheritance. It examined the new strategy of back-mutation prevention in the robot population.

Results:

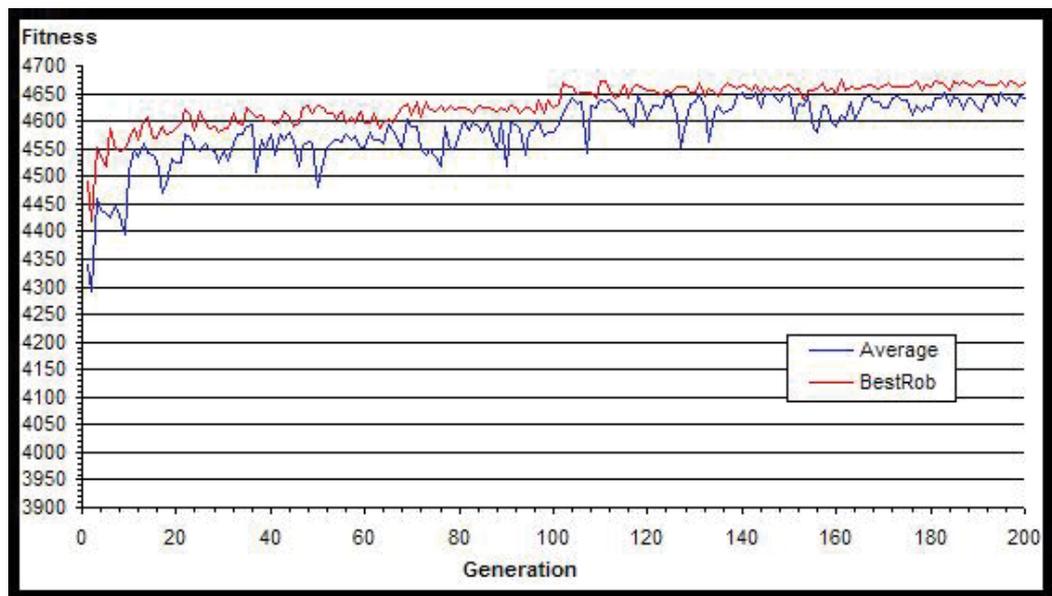


Figure 16. Evolution of the *neural network controller* using *back mutation prevention*, *inheritance selection*, *mutation rate of 3%*, and *generation time of 60s*. The population was randomly initialised. *BestRob* is the fitness of the best robot of the current generation and *Average* is the average fitness of all robots in the generation

Figure 16 shows a small increase in performance in comparison to experiment 2. The techniques developed so far succeeded in allowing the employment of the evolutionary system to control such a small population of robots.

Preventing back mutation was efficient in speeding up the process of finding a solution. Evolution was able to get close to maximum performance in less than 120 generations and was able to maintain a good result in the population thereafter. These results allow the evolution of a real system in 2 hours. The combined strategies of preventing back mutations and inheritance made viable the use of evolution to autonomously design robot controllers.

5. Conclusion

This work intended to provide understanding on the implementation of real evolutionary systems and inspiring insights that have potential of application in real time robotic control. When implementing an evolutionary system with real robots, the stochastic noise arising from the interactions of real physical systems makes very difficult to distinguish between global and local optima. The same robot can get different fitness values if evaluated again, even with the same configuration, due to noise on infrared signals, dust on the environment floor, etc. But the most important factor was the interaction between the robots. In this contest, good robots can get low fitness values if they are unlucky enough to spend most of the generation time navigating amongst poorly trained robots.

The suggested inheritance selection is a powerful strategy to prevent much of the chance factor from affecting the system. It succeeded in preventing most of the unfit individuals being mistaken as the best robot and protected the better robots from being erased by reproduction, giving them a chance to survive and repeat their better performance, until their average performance overcame the one of the current best robot.

The strategy of back-mutation prevention tested in the experiments was very efficient in looking for a solution in a considerably large search space. It demonstrated the power of the developed evolutionary system for many other applications of genetic algorithms, not only in robotics. The performance of the system was improved since the search space was reduced by using a neural network to implement a structured evolving controller.

6. References

- Baldassarre, G., Nolfi, S., & Parisi, D. (2003). Evolution of Collective Behavior in a Team of Physically Linked Robots, *Proceedings of the Second European Workshop on Evolutionary Robotics*, pp. 581-592, ISBN 3-540-00976-0, Essex, UK, April 14-16, 2003, Springer Verlag, Berlin
- Barker, w & Tyrrell. M. (2005). Hardware Fault-Tolerance Within the POETic System, *Proceedings of the 6th International Conference on Evolvable Systems*. LNCS 3637, pp. 25-36, ISBN: 3-540-28736-1, Barcelona, Spain, September, 2005, Springer-Verlag
- Bekey, G. A. (2005). *Autonomous Robots. From Biological Inspiration to Implementation and Control*. MIT Press. ISBN: 0-262-02578-7
- Burke, E., Gustafson, S., & Kendall, G. (2004). Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, Vol. 8, No. 1, 2004, pp. 47-62, ISSN: 1089-778X

- Ficici, S. G. & Pollack, J. B. (2000) Effects of Finite Populations on Evolutionary Stable Strategies. *Proceedings of the 2000 Genetic and Evolutionary Computation Conference*, pp. 927-934, ISBN: 1-55860-708-0, Las Vegas, Nevada, USA, Morgan-Kaufmann, 2000.
- Korenek, J. & Sekanina, L. (2005). Intrinsic evolution of sorting networks: a novel complete hardware implementation for FPGAs, *Proceedings of the 6th International Conference on Evolvable Systems*. LNCS 3637, pp. 46-55, ISBN: 3-540-28736-1, Barcelona, Spain, September, 2005, Springer-Verlag
- Liu, S., Tian, Y., & Liu, J. (2004). Multi robot path planning based on genetic algorithm. *Proceedings of 5th World Congress on Intelligent Control and Automation*, pp. 4706-4709, ISBN: 0-7803-8273-0, Hangzhou, China, June 2004
- Ludermir, T., Carvalho, A., Brage, A., & Souto, M. (1999). Weightless Neural Models: a Review of Current and Past Works. *Neural Computing Surveys*, v. 2, pp. 41-61, ISBN:1093-7609
- Michel, O. (2004). Webot: Professional mobile robot simulation. *Journal of Advanced Robotic Systems*, Vol. 1, No. 1, (March 2004), pp. 39-42, ISSN 1729-8806
- Mondada, F., D., & Nolfi, S. (2001). Co-Evolution and Ontogenetic Change in Competing Robots. In *Advances in the Evolutionary Synthesis of Intelligent Agents*, Publisher: MIT Press, Cambridge, MA, USA, ISBN: 0-262-16201-6, 30p., 2001.
- Nelson, A. L., Grant, E., Galeotti, J., & Rhody, S. (2004a). Maze exploration behaviors using an integrated evolutionary robotics environment. *Robotics and Autonomous Systems*, Vol. 46, N. 3, pp. 159-173, ISSN : 0921-8890
- Nelson, A. L., Grant, E., & Henderson, T. C. (2004b). Evolution of neural controllers for competitive game playing with teams of mobile robots. *Robotics and Autonomous Systems*, Vol. 46, N. 3, pp. 135-150, ISSN : 0921-8890
- Parker, C. A., Zhang, H., & Kube, C. R. (2003). Blind Bulldozing: Multiple Robot Nest Construction. *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2010-2015, ISBN-10: 0780378601, Las Vegas, Nevada, October 27-31, 2003, IEEE Press
- Parker L. & Touzet. C. (2000). Multi-robot learning in a cooperative observation task. *Proceedings of the 5th International Symposium on Distributed Autonomous Robotic Systems, DARS 2000*, pp. 391 – 402, ISBN 4-431-70295-4, Knoxville, Tennessee, USA, October 4-6, 2000, Springer,.
- Pollack, J. B., Lipson, H., Ficici, S. G., Funes, P., Hornby, G. S., & Watson, R. A. (2000). Evolutionary Techniques in Physical Robotics. *Proceedings of the Third International Conference on Evolvable Systems - ICES 2000: From Biology to Hardware (Lecture Notes in Computer Science; V. 1801)*, pp. 175-186, ISBN-10: 3540673385, April 17-19, 2000, Edinburgh, UK, Springer
- Seth, A. K. (1997). Interaction, Uncertainty, and the Evolution of Complexity. *Proceedings of the Fourth European Conference on Artificial Life*, Husbands, pp. 521-530, ISBN: 0262581574, Brighton UK, July 1997, P. and Harvey, I. (Eds.), MIT Press, 1997.
- Shipman, R., Shackleton, M., Ebner, M., & Watson, R. A. (2000). Neutral Search Spaces for Artificial Evolution: A Lesson From Life. *Proceedings of the Seventh International Workshop on the Synthesis and Simulation of Living Systems - Artificial Life VII*, pp. 52-59, ISBN-10: 026252290X, Reed College, Portland, Oregon, USA, Aug. 1-2, 2000

- Terrile, R. J., et al. (2005) Evolutionary Computation technologies for space systems. *Proceedings of the IEEE Aerospace Conference*, pp. 4284- 4295, ISBN: 0-7803-8870-4, Big Sky, March 2005.
- Thakoor, S., Morookian, J. M., Chahl, J., Hine, B., & Zormetzer, S. (2004). Bees: Exploring mars with bioinspired technologies. *Computer*, Vol. 37, No. 9, (Sept. 2004) pp. 38-47, ISSN: 0018-9162
- Tomassini, M. (1995). A Survey of Genetic Algorithms. *In Annual Reviews of Computational Physics*, Vol. III, 87-118, World Scientific, 1995, ISBN-10: 9810221762

Optimization of a 2 DOF Micro Parallel Robot Using Genetic Algorithms

Sergiu-Dan Stan, Vistrian Mătieș and Radu Bălan
*Technical University of Cluj-Napoca
Romania*

1. Introduction

Over the last couple of decades parallel robots have been increasingly studied and developed from both a theoretical viewpoint as well as for practical applications (Merlet, 1995). Advances in computer technology and development of sophisticated control techniques have allowed for the more recent practical implementation of parallel manipulators. Some of the advantages offered by parallel manipulators, when properly designed, include an excellent load-to-weight ratio, high stiffness and positioning accuracy and good dynamic behavior (Merlet 1995, Stan 2003). The ever-increasing number of publications dedicated to parallel robots illustrated very well this trend.

However, there are also some disadvantages associated with parallel manipulators, which have inhibited their application in some cases. Most serious of these is that the particular architecture of parallel manipulators leads to smaller manipulator workspaces than their serial counterparts.

One of the main influential factors on the performance of the micro parallel robot is its structural configuration. The kinematic relations, statics, dynamics and structure stiffness are all dependent upon it. After its choice, the next step on the manipulator design should be to establish its dimensions. Usually this dimensioning task involves the choice of a set of parameters that define the mechanical structure of the parallel robots. The parameter values should be chosen in a way to optimize some performance criterion, dependent upon the foreseen application. Micro parallel robots can also be difficult to design (Stan, 2003), since the relationships between design parameters and the workspace, and behavior of the manipulator throughout the workspace, are not intuitive by any means. This is one of the reasons why Merlet (1995) argues that customization of parallel manipulators for each application is absolutely necessary in order to ensure that all performance requirements can be met by the manipulator. As a result, development of design methodologies for such manipulators is an important issue in order to ensure performance to their full potential. In particular, the development and refinement of numerical methods for workspace determination of various parallel manipulators is of utmost importance.

There is a strong and complex link between the type of robot's geometrical parameters and its performance. It's very difficult to choose the geometrical parameters intuitively in such a way as to optimize the performance. Several papers have dealt with parallel robots to optimize performances (Stan, 2006). For example, various methods to determine workspace

of a parallel robot have been proposed using geometric or numerical approaches. Early investigations of robot workspace were reported by Gosselin (1990), Merlet (1994) and Cecarelli (2004). Stan (2003) presented a genetic algorithm approach for multi-criteria optimization of PKM. Most of the numerical methods to determine workspace of parallel manipulators rest on the discretization of the pose parameters in order to determine the workspace boundary. A method was proposed to determine the workspace by using optimization (Stan, 2006).

In the next sections, the planar 2-dof micro parallel robot of interest, and the kinematics for this manipulator, is presented. The 2-dof micro parallel robot considered in this study is shown in Fig. 3, where its joints (A and C) connected to the ground are active and the others are passive joints. The input motions of the active joints can be independent from each other or be provided via a set of gears maintaining a specified phase angle between the two active joints.

The objective of this chapter is to propose an optimization method for a planar micro parallel robot that uses performance evaluation criteria related to the workspace of micro parallel robot. Furthermore, a genetic algorithm is proposed as the principle optimization tool. The success of this type of algorithm for parallel robots optimization has been demonstrated in various papers (Stan, 2006).

2. Genetic Algorithms for Optimization of Micro Parallel Robots

2.1 Optimization based on Genetic Algorithms

Optimization is the process of making something better. An engineer or scientist conjures up a new idea and optimization improves on that idea. Optimization consists in trying variations on an initial concept and using the information gained to improve on the idea. Optimization is the process of adjusting the inputs to or characteristics of a device, mathematical process, or experiment to find the minimum or maximum output or result (Fig. 1). The input consists of variables, the process or function is known as the cost function, objective function, or fitness function, and the output is the cost or fitness. If the process is an experiment, then the variables are physical inputs to the experiment.

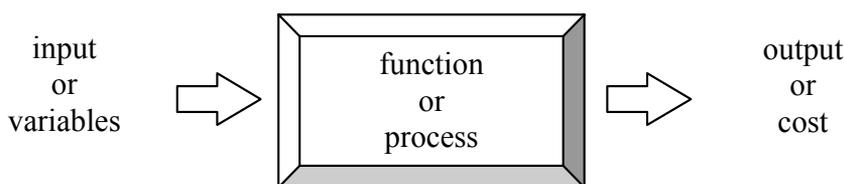


Figure 1. Diagram of a function or process that is to be optimized. Optimization varies the input to achieve a desired output

The genetic algorithm (GA) has been growing in popularity over the last few years as more and more researchers discover the benefits of its adaptive search. Genetic algorithms (GAs) were invented by John Holland in the 1960s and were developed by Holland and his students and colleagues at the University of Michigan in the 1960s and the 1970s. In contrast with evolution strategies and evolutionary programming, Holland's original goal was not to design algorithms to solve specific problems, but rather to formally study the phenomenon

of adaptation as it occurs in nature and to develop ways in which the mechanisms of natural adaptation might be imported into computer systems.

Holland's 1975 book *Adaptation in Natural and Artificial Systems* presented the genetic algorithm as an abstraction of biological evolution and gave a theoretical framework for adaptation under the GA. Holland's GA is a method for moving from one population of "chromosomes" (e.g., strings of ones and zeros, or "bits") to a new population by using a kind of "natural selection" together with the genetics-inspired operators of crossover, mutation, and inversion. Each chromosome consists of "genes" (e.g., bits), each gene being an instance of a particular "allele" (e.g., 0 or 1).

The selection operator chooses those chromosomes in the population that will be allowed to reproduce, and on average the fitter chromosomes produce more offspring than the less fit ones. Crossover exchanges subparts of two chromosomes, roughly mimicking biological recombination between two single-chromosome ("haploid") organisms; mutation randomly changes the allele values of some locations in the chromosome; and inversion reverses the order of a contiguous section of the chromosome, thus rearranging the order in which genes are arrayed. (Here, as in most of the GA literature, "crossover" and "recombination" will mean the same thing.)

In a broader usage of the term, a genetic algorithm is any population-based model that uses selection and recombination operators to generate new sample points in a search space. Many genetic algorithm models have been introduced by researchers largely working from an experimental perspective. Many of these researchers are application oriented and are typically interested in genetic algorithms as optimization tools. A constrained single-objective optimization problem can be converted to an unconstrained single-objective form by a penalty method (see, for example, Papalambros and Wilde, 1988).

Fig. 2 shows graphically how the most basic Genetic Algorithms operations are performed. The top block of binary numbers in Fig. 2 represents a population. Each row is an individual that represents one solution to the design problem. One individual is made up of all of the design variables concatenated. The GA user can decide how many binary digits are needed to represent each design variable.

In Fig. 2, there are 3 design variables of length 6, 7, and 4 binary digits. Initially, the population is generated randomly, and then the solutions are ranked from best to worst and a specified number of the lowest ranked individuals are replaced with combinations of the highest ranked individuals. The process of determining which of the highest ranked individuals are to be used is called selection. There are several differing methods of selection that can be used. Once selected, two individuals go through a process called crossover.

The crossover operation is also shown in Fig. 2, wherein two individuals (or parents) exchange a segment of the binary digits creating two new individuals (or offspring). Another basic Genetic A operation is mutation. Mutation can occur on the two individuals selected for crossover or on a single individual. The mutation operation randomly mutates or changes the bits within the individual based on the mutation probability set by the user.

Finding optimal values for the parameters used in the GA operations can be problematic. Parameter values that result in a relatively fast convergence for one problem may be slow for another. However, some general guidelines on the population size can be made on the basis of the binary string length of an individual.

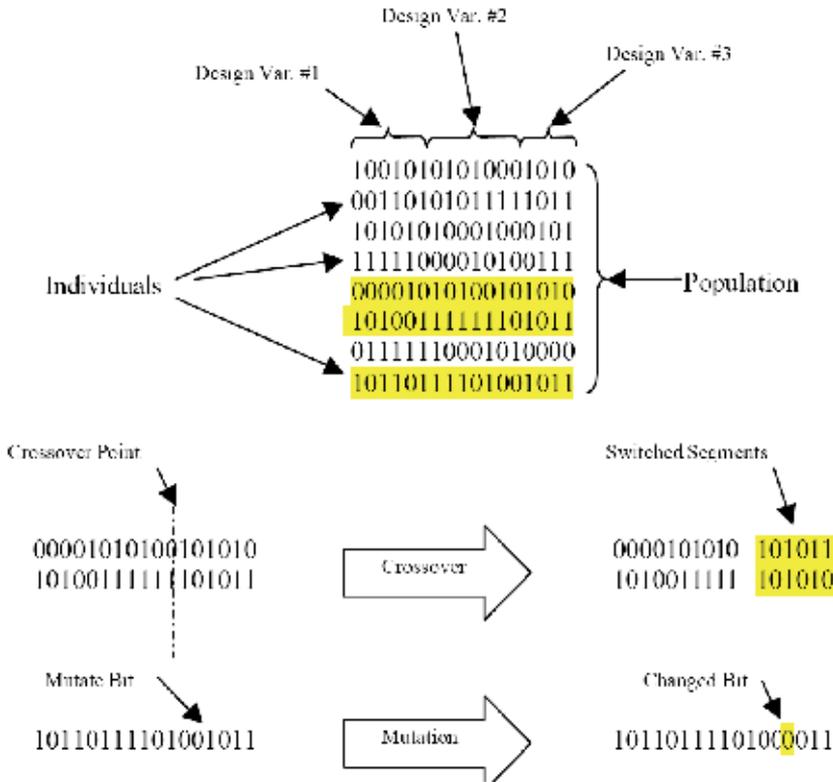


Figure 2. Graphical Description of Genetic Algorithms

Some of the advantages of a GA include that it:

- optimizes with continuous or discrete variables,
- doesn't require derivative information,
- simultaneously searches from a wide sampling of the cost surface,
- deals with a large number of variables,
- is well suited for parallel computers,
- optimizes variables with extremely complex cost surfaces (they can jump out of a local minimum),
- provides a list of optimum variables, not just a single solution,
- may encode the variables so that the optimization is done with the encoded variables,
- works with numerically generated data, experimental data, or analytical functions.

These advantages are intriguing and produce stunning results when traditional optimization approaches fail miserably.

2.2 Pareto-optimality

Multiobjective problems are special in the sense that they do not have a unique solution. Usually there is no single solution for which all objectives are optimal. The solution to a multiobjective problem therefore comprises a *set* of solutions for which holds that there are no other solutions that are superior considering *all* objectives. These solutions are called

Pareto-optimal. Hence, optimizing a multiobjective problem is comprised of finding Pareto optimal solutions.

The notion of Pareto-optimality is defined in terms of *dominance*. Let's assume that a multiobjective problem has k objectives. Assuming that this is a minimization problem, then a solution $x = (x_1, x_2, \dots, x_k)$ is said to dominate another solution $y = (y_1, y_2, \dots, y_k)$ if $\forall i x_i \leq y_i$ and $\exists i x_i < y_i$. Solution x is a member of the Pareto-set, or said to be *non-dominated*, if there is no other solution y such that y dominates x . The multiobjective problem can now be defined as finding solutions which are non-dominated.

Over the years, several evolutionary approaches to multiobjective problems have been introduced. The most commonly used approach is to combine the objective function into a single objective function using weighting coefficients and penalty functions. This problem-transformation enables the use of a simple single-objective genetic algorithm to find a single solution, which may be feasible, so requiring no further searches. Weights and penalty functions are generally hard to set accurately though, whereas both are very problem dependent (Richardson et al., 1989). As a result, the solution a GA comes up with may not fulfill all the designer's needs. The solution may not even be non-dominated. Setting the weights correctly requires a certain amount of search space knowledge, which is often not available in advance. This way of dealing with multiobjective optimization is therefore not always applicable or efficient.

Another, and perhaps more effective approach, is to use genetic algorithms to locate Pareto-optimal solutions. These solutions are, by definition, located on a boundary, known as the Pareto-front. We would like the solutions to cover the Pareto-front as well as possible, as to obtain a good representation of this front. This approach requires an extensive exploration of the search space and it is this requirement that makes evolutionary algorithms extremely applicable in this case. Their massive parallel exploration of search spaces is an invaluable advantage over other more conventional techniques in locating Pareto-optimal solutions. This Pareto-based approach has additional benefits as well. This approach offers multiple solutions from which a decision maker can select the solution that is best suited according to additional criteria, without requiring additional searches. Pareto based optimization is hence a more transparent and efficient way of dealing with multiobjective problem.

A Pareto solution is proper if the tradeoff rate between the objectives in the neighborhood of that solution is bounded. In other words, in the neighborhood of a proper Pareto point, a finite increase in one objective is possible only at the expense of some reasonable decrease in one other objective. A proper Pareto solution is a good candidate design solution (Stan, 2003).

3. Two DOF RRRRR Micro Parallel Robot

3.1 Geometrical description of the micro parallel robot

The micro parallel robot considered in this paper is the 2-dof planar parallel mechanism shown in Fig. 3. The micro parallel robot consists of a five-bar mechanism connected to a base by two rotary actuators, which control the two output degrees of freedom of the end-effector. The actuators are joined to the base and platform by means of revolute joints identified by the letters A-D. It will be assumed that $AO=OC=AC/2$. The coordinates of point P, the end-effector point, are (x_P, y_P) . In more general terms, the actuator joint angles are the input variables, i.e. $\mathbf{v}=[q_1, q_2]^T \in \mathbb{R}^2$. The global coordinates of the working point P

form the output coordinates, i.e. $\mathbf{u}=[x_p, y_p]^T \in \mathbb{R}^2$. The 2-dof micro parallel robot may be used only for positioning P in the x - y plane. It is evident that this manipulator thus has 2-dof. Thus, the generalized coordinates for this kind of micro parallel robot are therefore given by:

$$\mathbf{q}=[\mathbf{u}^T, \mathbf{v}^T]^T=[x_p, y_p, q_1, q_2]^T \in \mathbb{R}^4 \quad (1)$$

In general, factors imposed by the physical construction of the planar parallel manipulator, which limit the workspace, may be related to the input variables or a combination of input, output and intermediate variables. An example of former type for the planar parallel manipulator are joint angles limits, and of the latter, limits on the angular displacement of the revolute joints connecting the legs to the ground and to the platform. These limiting factors are described by means of inequality constraints and may, respectively, take the general forms:

$$\mathbf{v}^{\min} \leq \mathbf{v} \leq \mathbf{v}^{\max} \quad (2)$$

$$\mathbf{g}^{\min} \leq \mathbf{g}(\mathbf{u}, \mathbf{v}) \leq \mathbf{g}^{\max} \quad (3)$$

The above general definitions are necessary in order to facilitate the mathematical description of kinematics and workspaces of the 2-dof planar micro parallel robot. In this study mixed constraints, represented by (3), are not taken into consideration.

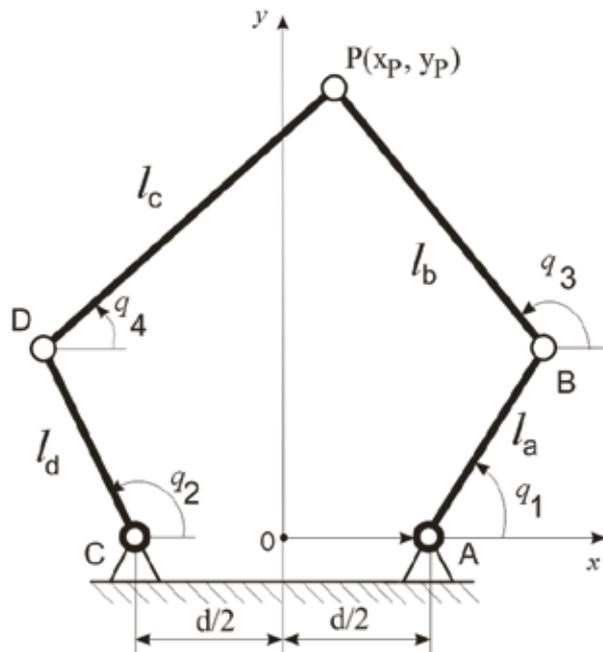


Figure 3. Micro parallel robot with 2 degrees of freedom

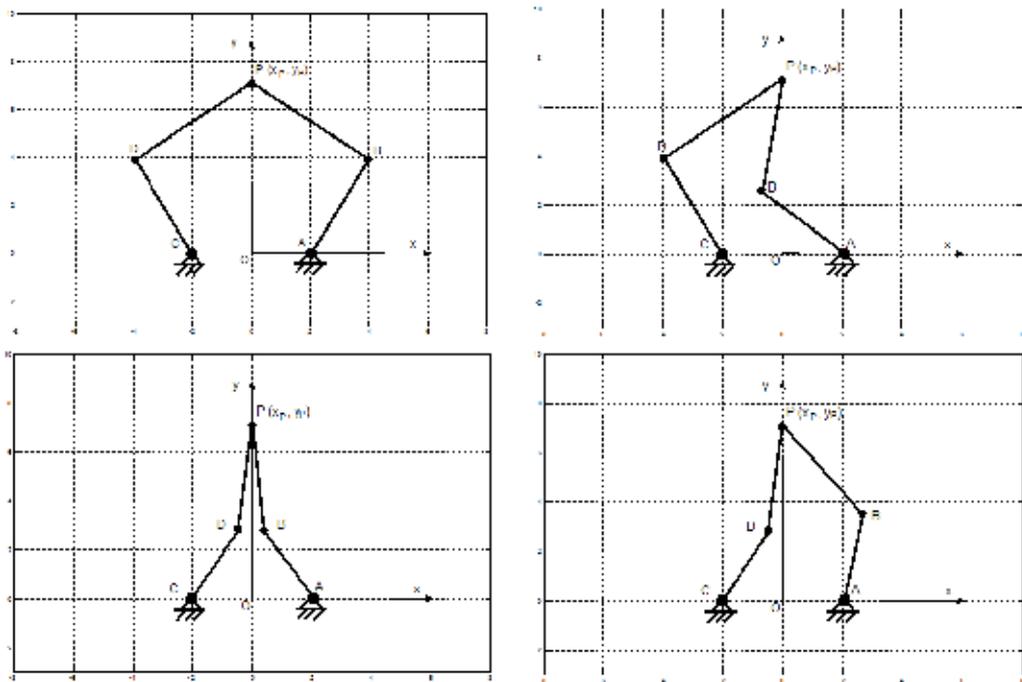


Figure 4. The 4 types of functioning of micro parallel robot with 2 degrees of freedom

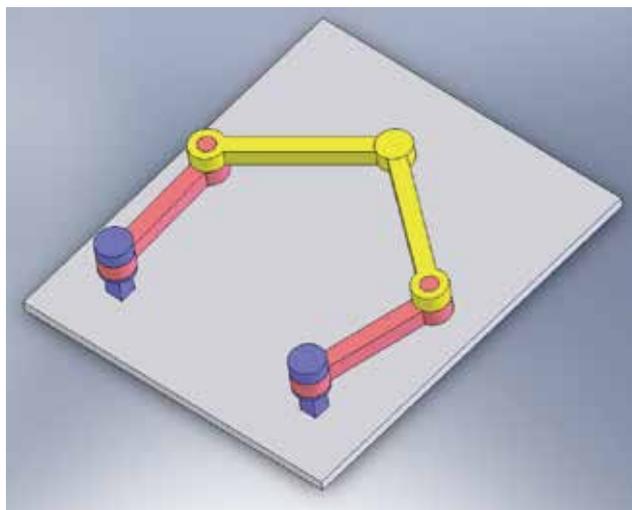


Figure 5. CAD model of the micro parallel robot of type RRRRR

3.2 The kinematics of the RRRRR micro parallel robot

Kinematic analysis of five-bar micro parallel robot is needed before carrying out derivations for the mathematical model. It is considered the five-bar micro parallel robot with revolute joints as in Fig. 1. It is known the length of the links as well as the fixed joints coordinates. The five-bar mechanism is symmetric toward Oy -axis, thus $l_a = l_d = l$ respectively $l_b = l_c = L$.

Actuators are placed in A and C. Attaching to each link a vector, on the $OABPO$ respectively $OCDPO$, we can write successively the relations:

$$\overrightarrow{OP} = \overrightarrow{OA} + \overrightarrow{AB} + \overrightarrow{BP}; \overrightarrow{OP} = \overrightarrow{OC} + \overrightarrow{CD} + \overrightarrow{DP} \quad (4)$$

Based on the above relations, the coordinates of the point P have the following forms:

$$\begin{aligned} x_P &= \frac{d}{2} + l \cos q_1 + L \cos q_3 = -\frac{d}{2} + l \cos q_2 + L \cos q_4 \\ y_P &= l \sin q_1 + L \sin q_3 = l \sin q_2 + L \sin q_4 \end{aligned} \quad (5)$$

In this part, kinematics of a planar micro parallel robot articulated with revolute type joints has been formulated to solve direct kinematics problem, where the position, velocity and acceleration of the micro parallel robot end-effector are required for a given set of joint position, velocity and acceleration.

The Direct Kinematic Problem (DKP) of micro parallel robot is an important research direction of mechanics, which is also the most basic task of mechanic movement analysis and the base such as mechanism velocity, mechanism acceleration, force analysis, error analysis, workspace analysis, dynamical analysis and mechanical integration. For this kind of micro parallel robot solving DKP is easy. Coordinates of point P in the case when values of joint angles are known q_1 and q_2 are obtained from relations:

$$x_P = \frac{-D \pm \sqrt{D^2 - 4BC}}{2C}, \quad y_P = \frac{A - x_P(x_B - x_D)}{y_B - y_D} \quad (6)$$

where:

$$\begin{aligned} A &= -\frac{1}{2}(x_D^2 + y_D^2 - x_B^2 - y_B^2 - L_{DP}^2 + L_{BP}^2) \\ B &= (y_B - y_D)^2(x_D^2 + y_D^2 - L_{DP}^2) + A^2 - 2y_D(y_B - y_D)A \\ C &= (y_B - y_D)^2 + (x_B - x_D)^2 \\ D &= 2y_D(y_B - y_D)(x_B - x_D) - 2x_D(y_B - y_D)^2 - 2A(x_B - x_D) \\ x_D &= -\frac{d}{2} + l \cos q_2 & y_D &= l \sin q_2 \\ x_B &= \frac{d}{2} + l \cos q_1 & y_B &= l \sin q_1 \end{aligned} \quad (7)$$

The speed of the point P is obtained differentiating the relations (1). Thus results:

$$J_A \cdot \begin{bmatrix} V_x \\ V_y \end{bmatrix} = J_B \cdot \begin{bmatrix} \omega_1 \\ \omega_3 \end{bmatrix} \quad (8)$$

where

$$J_A = \begin{bmatrix} L \cos q_3 & L \sin q_3 \\ L \cos q_4 & L \sin q_4 \end{bmatrix} \quad (9)$$

$$J_B = \begin{bmatrix} l \cdot L \sin(q_1 - q_3) & 0 \\ 0 & l \cdot L \sin(q_2 - q_4) \end{bmatrix} \quad (10)$$

or

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = J \cdot \begin{bmatrix} \omega_1 \\ \omega_3 \end{bmatrix} \quad (11)$$

where

$$J = J_B J_A^{-1} = \frac{L}{\sin(q_4 - q_3)} \begin{pmatrix} \sin q_4 \sin(q_1 - q_3) & -\sin q_3 \sin(q_2 - q_4) \\ -\cos q_4 \sin(q_1 - q_3) & \cos q_3 \sin(q_2 - q_4) \end{pmatrix} \quad (12)$$

and J represents the Jacobian matrix.

Acceleration of the point P is obtained by differentiating of relation (8), as it yields:

$$\begin{bmatrix} A_x \\ A_y \end{bmatrix} = J \cdot \begin{bmatrix} \varepsilon_1 \\ \varepsilon_3 \end{bmatrix} + \frac{d}{dt} J \cdot \begin{bmatrix} \omega_1 \\ \omega_3 \end{bmatrix} \quad (13)$$

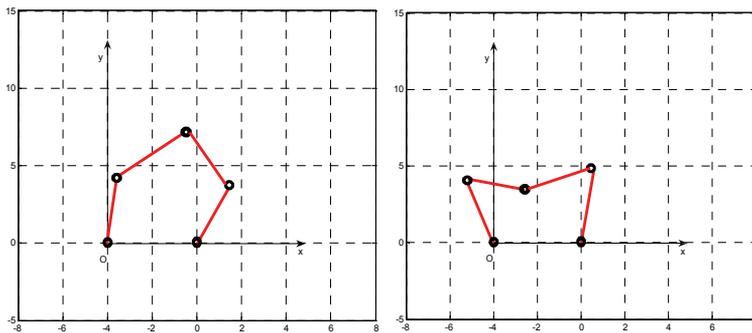


Figure 6. The two forward kinematic models: (a) the up-configuration and (b) the down-configuration

Based on the inverse kinematics analysis are determined the motion laws of the actuator links function of the kinematics parameters of point P .

The values of joint angles q_i , ($i = 1 \dots 4$) knowing the coordinates x_P, y_P of point P , may be computed with the following relations:

$$q_1 = 2 \operatorname{arctg} \left(\frac{-B + \sigma_i \sqrt{B^2 - (C^2 - A^2)}}{C - A} \right) \quad (14)$$

$$q_3 = \operatorname{arctg} \left(\frac{\sqrt{M^2 + N^2 - P^2}}{P} \right) + \operatorname{arctg} \left(\frac{N^2}{M^2} \right)$$

$$q_2 = 2 \operatorname{arctg} \left(\frac{-B + \sigma_i \sqrt{B^2 - (f^2 - e^2)}}{f - e} \right)$$

$$q_4 = 2 \operatorname{arctg} \left(\frac{-b \pm \sqrt{b^2 - (F^2 + E^2)}}{F - E} \right), \quad \sigma_i = 1 \text{ or } -1$$

where

$$A = -2l \left(x_P - \frac{d}{2} \right) \quad (15)$$

$$a = -2L \left(x_P - \frac{d}{2} \right)$$

$$C = \left(x_P - \frac{d}{2} \right)^2 + y_P^2 + l^2 - L^2$$

$$M = 2L\lambda$$

$$N = 2L\mu$$

$$b = -2y_P L$$

$$c = \left(x_P - \frac{d}{2} \right)^2 + y_P^2 - l^2 - L^2$$

$$e = -2l \left(x_P + \frac{d}{2} \right)$$

$$f = \left(x_P + \frac{d}{2} \right)^2 + y_P^2 + l^2 - L^2$$

$$B = -2y_P l$$

$$F = \left(x_p + \frac{d}{2}\right)^2 + y_p^2 - l^2 + L^2$$

$$E = -2L\left(x_p + \frac{d}{2}\right)$$

$$P = -\lambda^2 - \mu^2$$

$$\lambda = l \cdot \cos q_1 - l \cdot \cos q_2 + d$$

$$\mu = l \cdot \sin q_1 - l \cdot \sin q_2$$

From Eq. (14), one can see that there are four solutions for the inverse kinematics problem of the 2-dof micro parallel robot. These four inverse kinematics models correspond to four types of working modes (see Fig. 7).

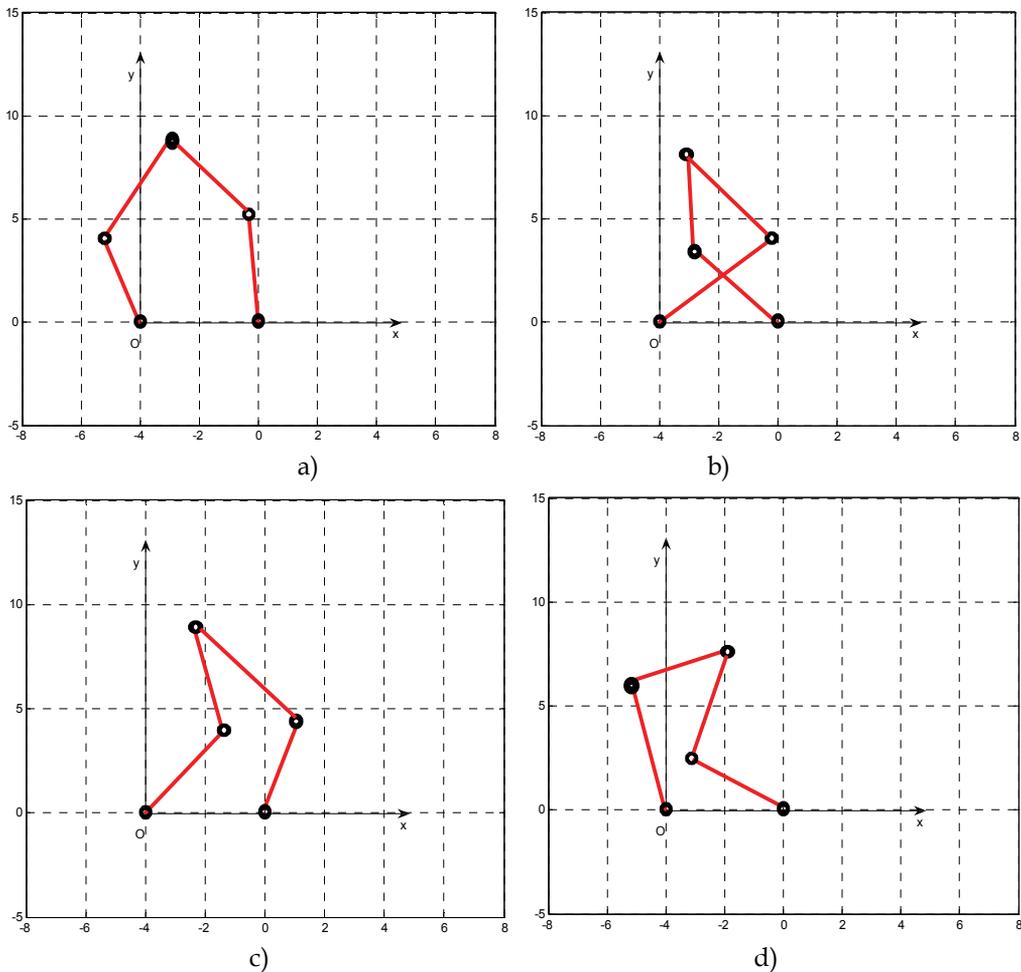


Figure 7. The four inverse kinematics models: (a)''+'' model; (b)''-+'' model; (b)''--'' model; (d)''++'' model



Figure 8. Graphical User Interface for solving the inverse kinematics problem of 2 DOF micro parallel robot

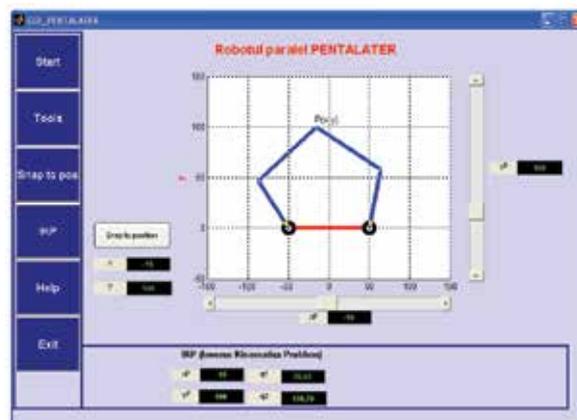


Figure 9. Robot configuration for micro parallel robot $x_p=-15$ mm $y_p=100$ mm

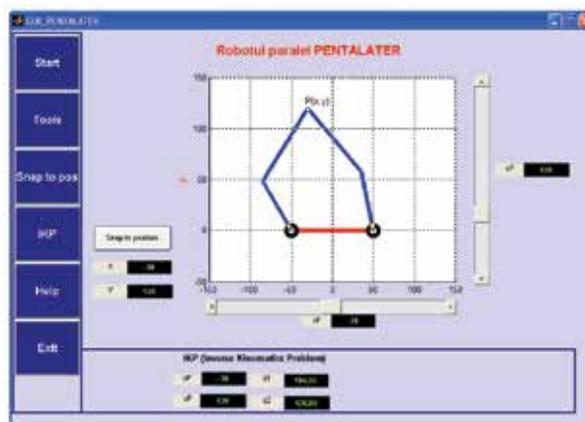


Figure 10. Robot configuration for micro parallel robot $x_p=-30$ mm $y_p=120$ mm

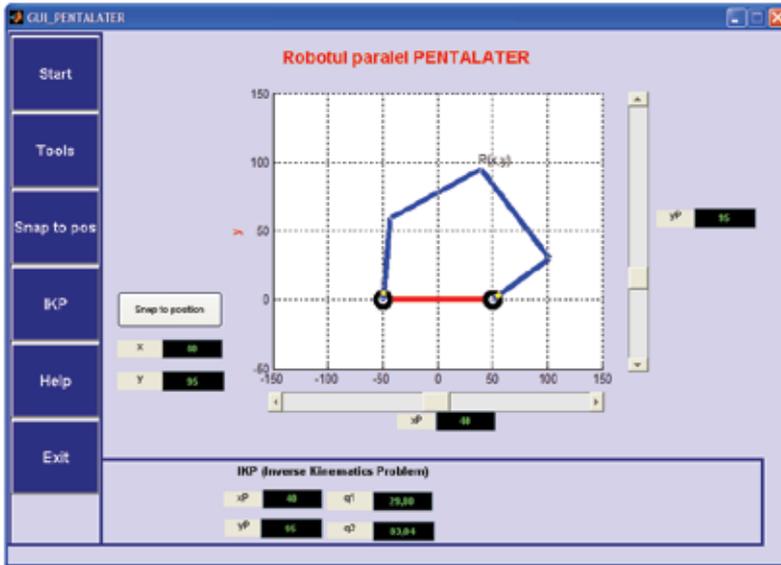


Figure 11. Robot configuration for micro parallel robot $x_p=40$ mm $y_p=95$ mm

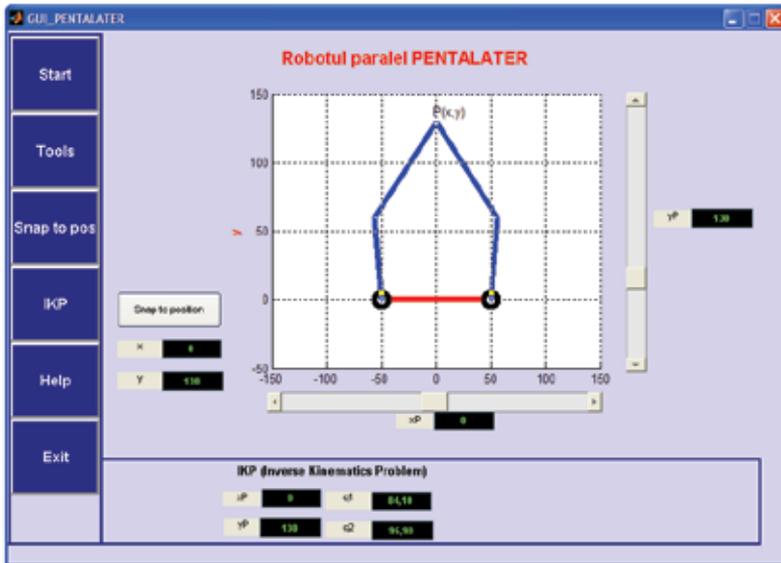


Figure 12. Robot configuration for micro parallel robot $x_p=0$ mm $y_p=130$ mm

3.3 Singularities analysis of the planar 2-dof micro parallel robot

In the followings, vector \mathbf{v} is used to denote the actuated joint coordinates of the manipulator, representing the vector of kinematic input. Moreover, vector \mathbf{u} denotes the Cartesian coordinates of the manipulator gripper, representing the kinematic output. The velocity equations of the micro parallel robot can be rewritten as:

$$A\dot{\mathbf{u}} + B\dot{\mathbf{v}} = 0 \tag{16}$$

Where $\dot{v} = [\dot{q}_1, \dot{q}_2]^T$, $\dot{u} = [\dot{x}_p, \dot{y}_p]^T$ and where A and B are square matrices of dimension 2, called Jacobian matrices, with 2 the number of degrees of freedom of the micro parallel robot. Referring to Eq. (13), (Gosselin and Angeles, 1990), has defined three types of singularities which occur in parallel kinematics machines.

(I) The first type of singularity occurs when $\det(B)=0$. These configurations correspond to a set of points defining the outer and internal boundaries of the workspace of the micro parallel robot.

(II) The second type of singularity occurs when $\det(A)=0$. This kind of singularity corresponds to a set of points within the workspace of the micro parallel robot.

(III) The third kind of singularity when the positioning equations degenerate. This kind of singularity is also referred to as an architecture singularity (Stan, 2003). This occurs when the five points ABCDP are collinear.

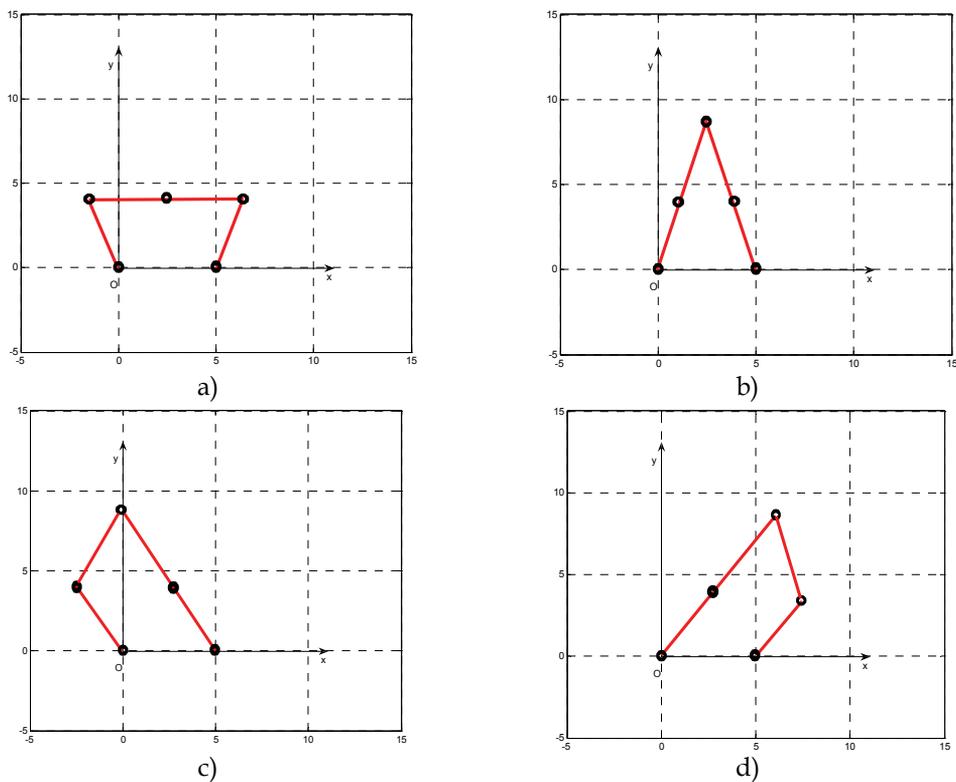


Figure 13. Some configurations of singularities: (a) the configuration when l_b and l_c are completely extended (b) both legs are completely extended; (c) the second leg is completely extended and (d) the first leg is completely extended

In this chapter, it will be used to analyze the second type of singularity of the 2-dof micro parallel robot introduced above in order to find the singular configuration with this type of micro parallel robot. For the first type of singularity, the singular configurations can be obtained by computing the boundary of the workspace of the micro parallel robot.

Furthermore, it is assumed that the third type of singularity is avoided by a proper choice of the kinematic parameters.

For this micro parallel robot, we can use the angular velocities of links l_c and l_b as the output vector. Matrix A is then written as:

$$A = \begin{bmatrix} l_c \cdot \cos(q_4) & -l_b \cdot \cos(q_3) \\ l_c \cdot \sin(q_4) & -l_b \cdot \sin(q_3) \end{bmatrix} \quad (17)$$

From Eq. (17), one then obtains:

$$\det(A) = l_c \cdot l_b \cdot \sin(q_4 - q_3). \quad (18)$$

From Eq. (18), it is clear that when $q_4 = q_3 + n\pi$, $n = 0, \pm 1, \pm 2, \dots$, then $\det(A) = 0$. In other words if the two links l_c and l_b are along the same line, the micro parallel robot is in a configuration which corresponds to be second type of singularity.

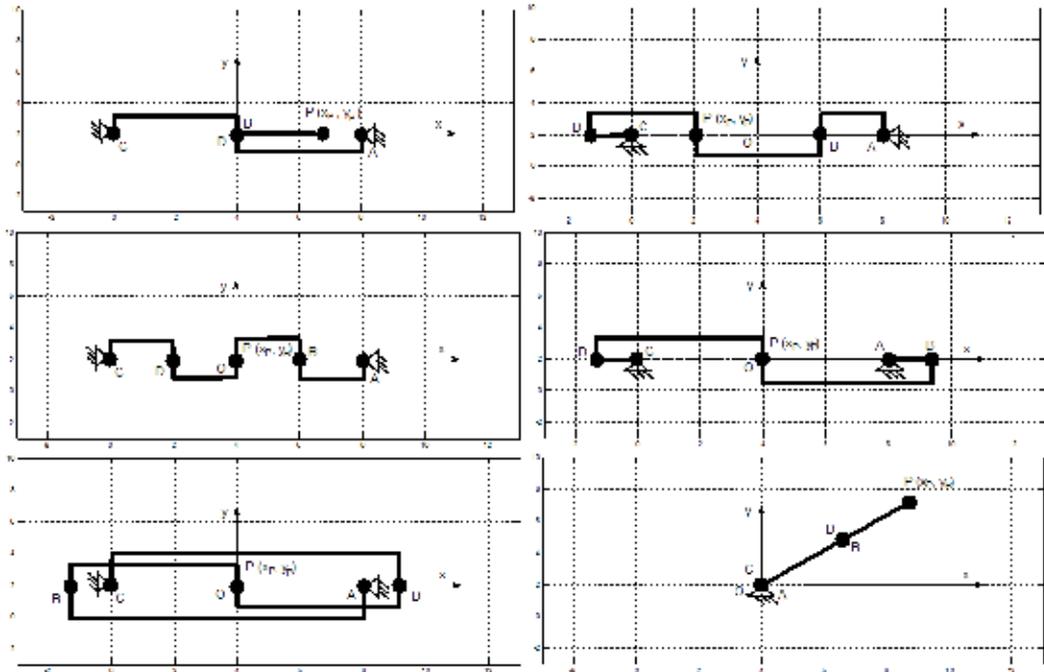


Figure 14. Examples of architectural singular configurations of the RRRRR micro parallel robot

3.4 Optimal design of the planar 2-dof micro parallel robot

The performance index chosen corresponds to the workspace of the micro parallel robot. Workspace is defined as the region that the output point P can reach if q_1 and q_2 changes from 2π without the consideration of interference between links and the singularities. There

were identified five types of workspace shapes for the 2-dof micro parallel robot as it can be seen in Figs. 15-20.

Each workspace is symmetric about the x and y axes. Workspace was determined using a program made in MATLAB™. Analysis, visualization of workspace is an important aspect of performance analysis. A numerical algorithm to generate reachable workspace of parallel manipulators is introduced.

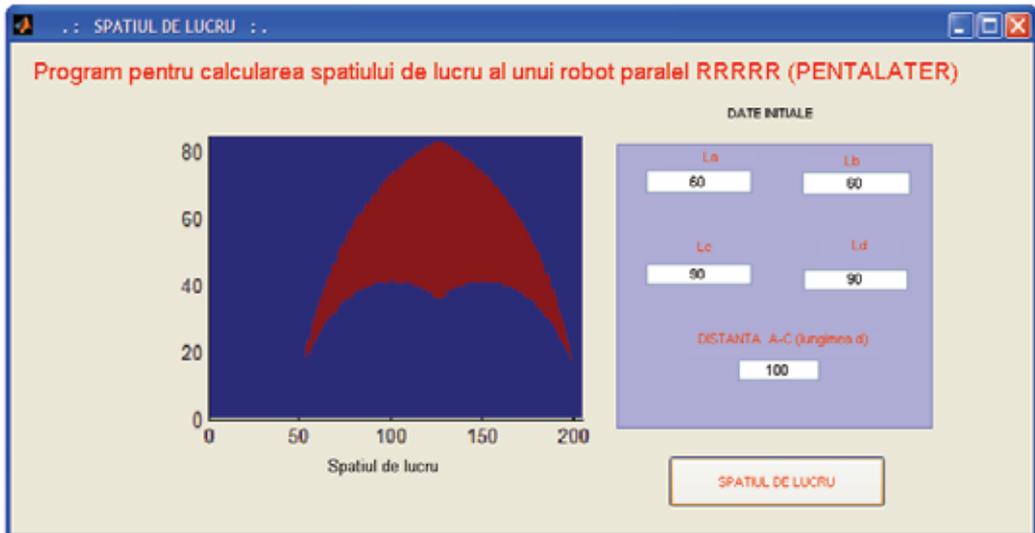


Figure 15. The GUI for calculus of workspace for the planar 2 DOF micro parallel robot

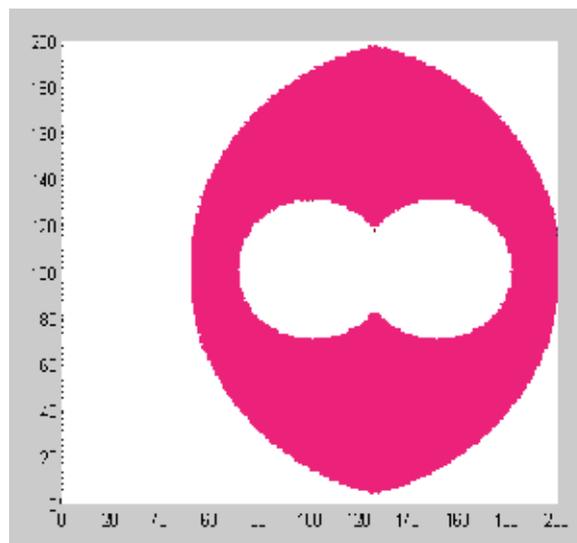


Figure 16. Workspace of the 2 DOF micro parallel robot

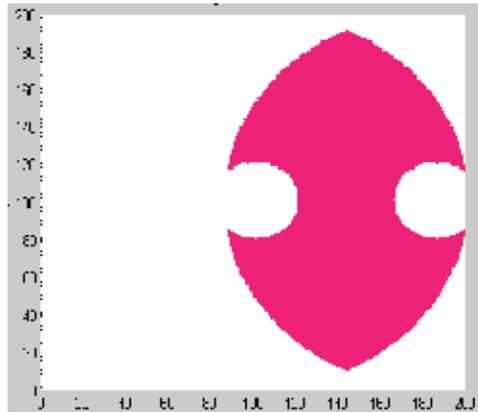


Figure 17. Workspace of the 2 DOF micro parallel robot

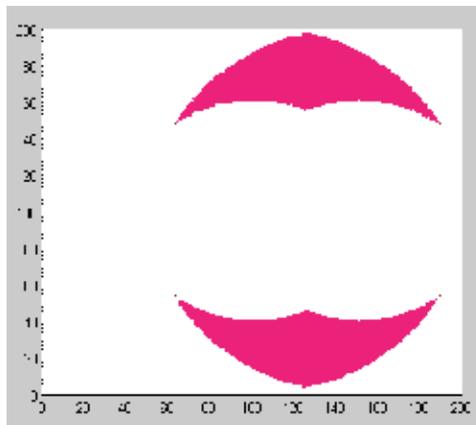


Figure 18. Workspace of the 2 DOF micro parallel robot

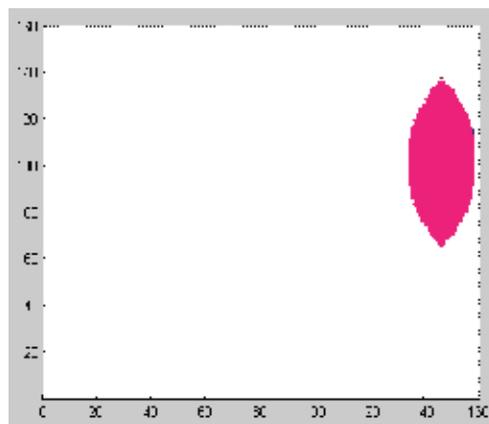


Figure 19. Workspace of the 2 DOF micro parallel robot

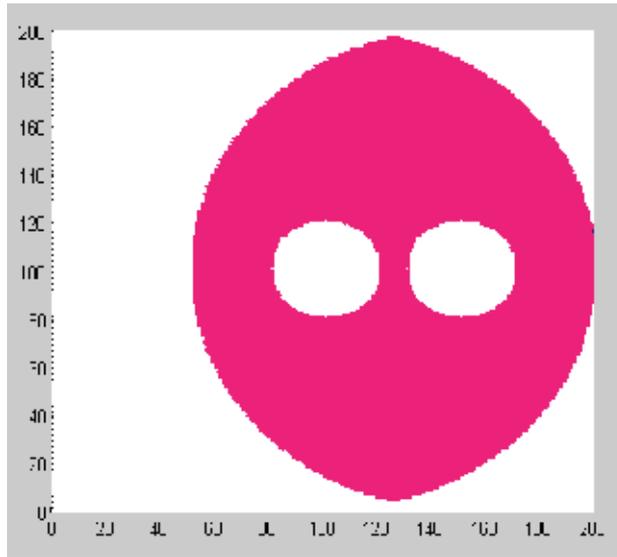


Figure 20. Workspace of the 2 DOF micro parallel robot

The above design of 2 DOF micro parallel robot employed mainly traditional optimization design methods. However, these traditional optimization methods have drawbacks in finding the global optimal solution, because it is so easy for these traditional methods to trap in local minimum points (Stan, 2003).

GA refers to global optimization technique based on natural selection and the genetic reproduction mechanism. GA is a directed random search technique that is widely applied in optimization problems. This is especially useful for complex optimization problems where the number of parameters is large and the analytical solutions are difficult to obtain. GA can help to find out the optimal solution globally over a domain.

The design of the micro parallel robot can be made based on any particular criterion. Here a genetic algorithm approach was used for workspace optimization of 2 DOF micro parallel robot.

For simplicity of the optimization calculus a symmetric design of the structure was chosen. In order to choose the robot dimensions d , l_a , l_b , l_c , l_d we need to define a performance index to be maximized. The chosen performance index is workspace W .

One objective function is defined and used in optimization. It is noted as W , and corresponds to the optimal workspace. We can formalize our design optimization problem as the following equation:

$$\text{Obj_function}=\max(W) \quad (19)$$

Optimization problem is formulated as follows: the objective is to evaluate optimal link lengths which maximize (16). The design variables or the optimization factor is the ratios of the minimum link lengths to the base link length b , and they are defined by:

$$l_i/d \quad (20)$$

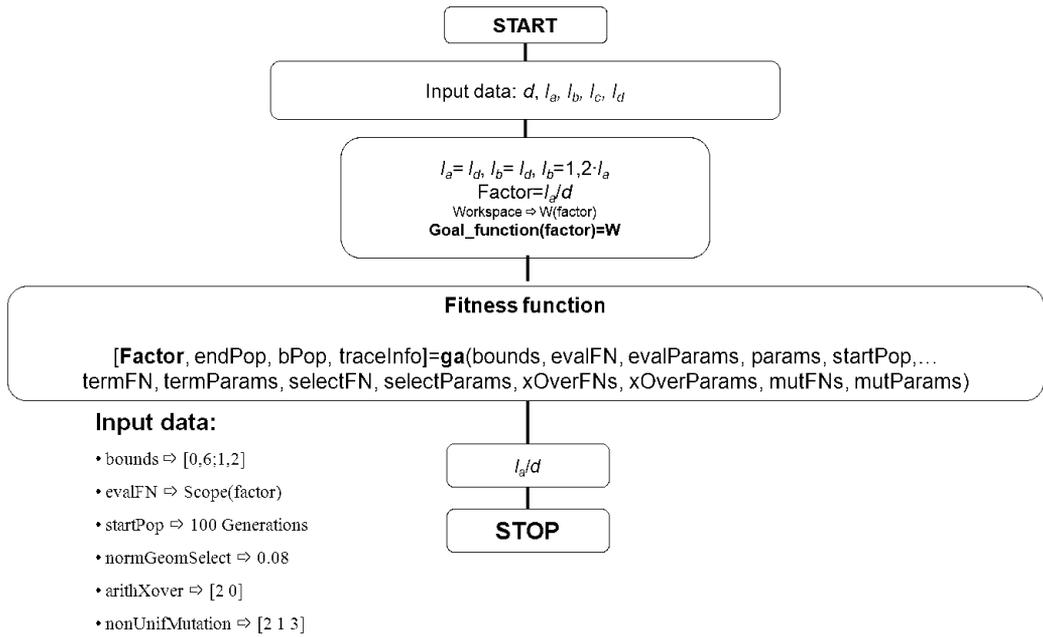


Figure 21. Flowchart of the optimization Algorithm with GAOT (Genetic Algorithm Optimization Toolbox)

Constraints to the design variables are:

$$0,6 < l_d/d < 1,2 \quad (21)$$

$$l_a = l_d, l_b = l_c, l_b = 1,2 l_a \quad (22)$$

For this example the lower limit of the constraint was chosen to fulfill the condition $l_d \geq d/2$. For simplicity of the optimization calculus the upper bound was chosen $l_d \leq 1,2d$.

During optimization process using genetic algorithm it was used the following GA parameters, presented in Table 1. A genetic algorithm (GA) is used because its robustness and good convergence properties. The GA approach has the clear advantage over conventional optimization approaches in that it allows a number of solutions to be examined in a single design cycle. The traditional methods searches optimal points from point to point, and are easy to fall into local optimal point. Using a population size of 50, the GA was run for 100 generations. A list of the best 50 individuals was continually maintained during the execution of the GA, allowing the final selection of solution to be made from the best structures found by the GA over all generations.

We performed a kinematic optimization in such a way to maximize the workspace index W . It is noticed that optimization result for micro parallel robot when the maximum workspace of the 2 DOF planar micro parallel robot is obtained for $l_d / d = 1,2$. The used dimensions for the 2 DOF parallel micro robot were: $l_a = 72$ mm, $l_b = 87$ mm, $l_c = 87$ mm, $l_d = 72$ mm, $d = 60$ mm. Maximum workspace of the micro parallel robot was found to be $W = 9386$ mm². The results show that GA can determine the architectural parameters of the robot that provide an optimized workspace. Since the workspace of a micro parallel robot is far from being intuitive, the method developed should be very useful as a design tool.

1	Population	50
2	Generations	100
3	Crossover rate	0,08
4	Mutation rate	0,005

Table 1. GA Parameters

However, in practice, optimization of the micro parallel robot geometrical parameters should not be performed only in terms of workspace maximization. Some parts of the workspace are more useful considering a specific application.

Indeed, the advantage of a bigger workspace can be completely lost if it leads to new collision in parts of it which are absolutely needed in the application. However, it's not the case of the presented structure.

In the second case of optimization of the 2 DOF micro parallel robot there have been used 4 optimization criteria:

1. *transmission quality index* $\rightarrow T=1$ the best value and the maximum one
2. *workspace* \rightarrow a higher value is desirable
3. *stiffness index* \rightarrow a higher value is desirable
4. *manipulability index* \rightarrow a higher value is desirable

Beside workspace which is an important design criterion, transmission quality index is another important criterion.

The transmission quality index couples velocity and force transmission properties of a parallel robot, i.e. power features (Hesselbach et al., 2003). Its definition runs:

$$T = \frac{\|I\|^2}{\|J\| \cdot \|J^{-1}\|} \quad (23)$$

where I is the unity matrix.

T is between $0 < T < 1$; $T=0$ characterizes a singular pose, the optimal value is $T=1$ which at the same time stands for isotropy (Hesselbach et al., 2003).

The manipulability condition number is a quality number in the sense of Yoshikawa, can be defined in terms of the ratio of a measure of performance in the task space and a measure of effort in the joint space.

$$M = \sqrt{\det(J \cdot J^T)} \quad (24)$$

If J is quadratic Eq. (4) reduces to $M=\det(J)$. The goal is to have a value of M as large as possible.

The stiffness condition number runs using the matrix K :

$$S = \|K^{-1}\| \cdot \|K\| = \|J \cdot J^T\| \cdot \|(J \cdot J^T)^{-1}\| \quad (25)$$

If the guiding chains of the machine between frame and working platform have different stiffness, the matrix K must be replaced by the matrix:

$$K_C = (J^T)^{-1} \cdot C \cdot J^{-1} \quad (26)$$

where the diagonal matrix C contains the stiffness of the single guiding chain. The reciprocal value of S is between $0 < 1/S \leq 1$; a singular pose is again characterized by $1/S = 0$, whereas $1/S = 1$ is the optimal (isotropic) index. In the following figures, the performances evaluation throughout the workspace of the planar 2 DOF micro parallel robot is presented.

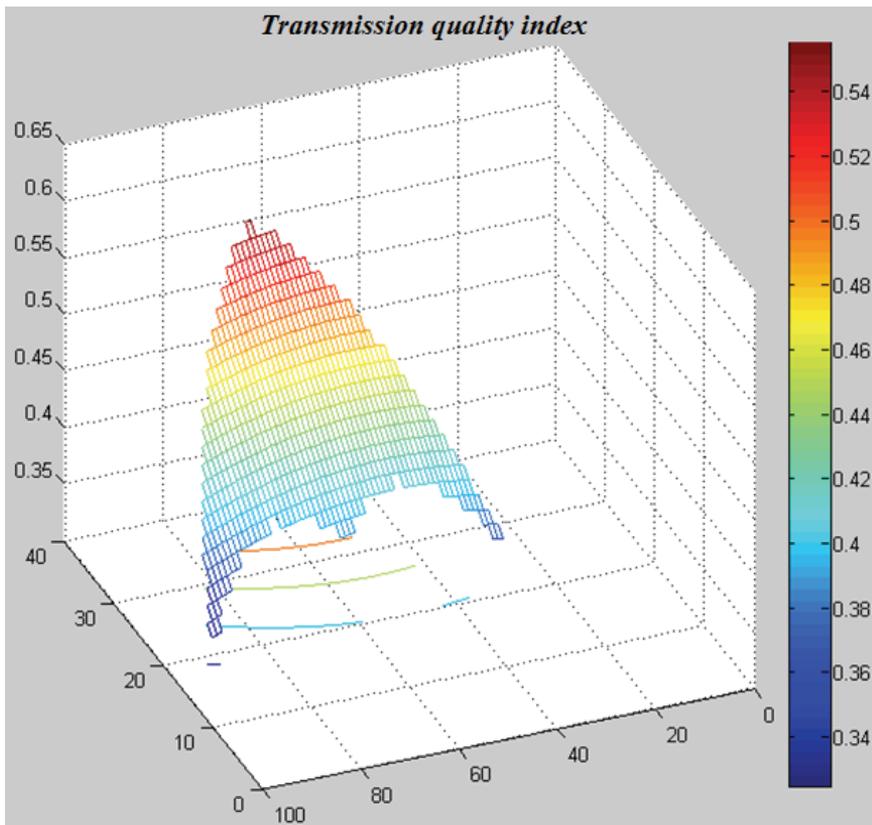


Figure 22. Transmission quality index for 2 DOF micro parallel robot

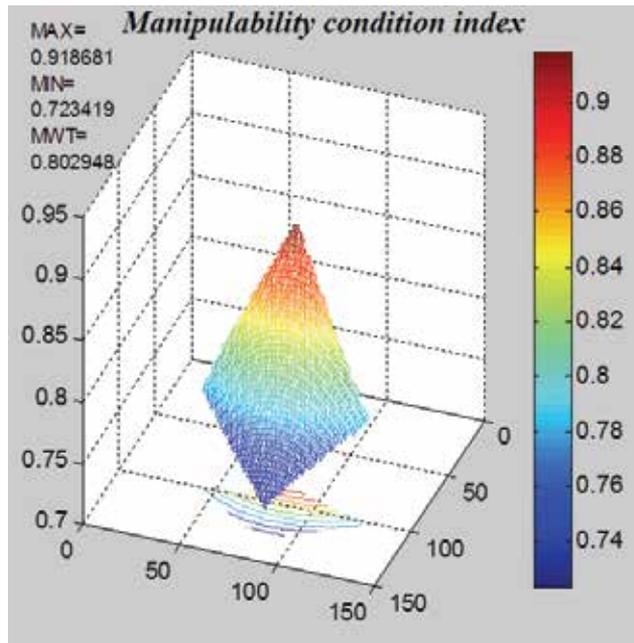


Figure 23. Manipulability index for 2 DOF micro parallel robot

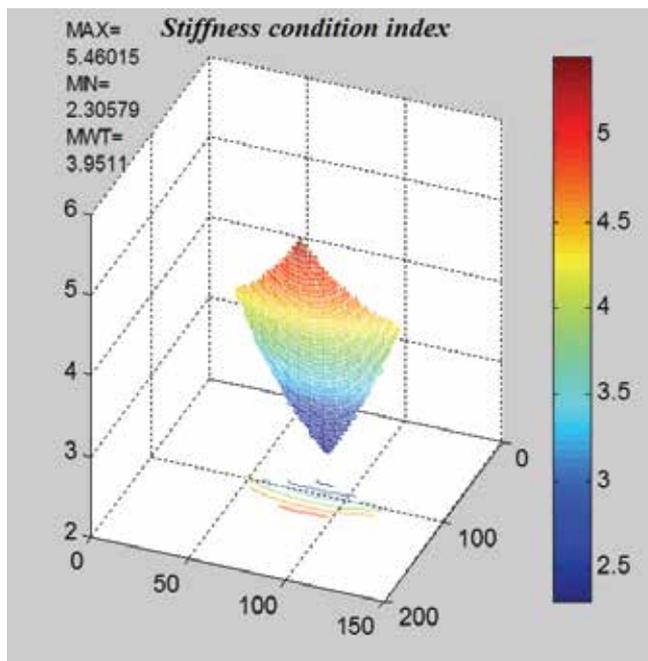


Figure 24. Stiffness index for 2 DOF micro parallel robot

Objective function:

$$\text{Obj_Fun} = f(T, A, S, M)$$

Optimization parameters:

$$l_d \text{ and } d$$

Constraints:

$$80 \leq l_d \leq 110 \text{ and } 90 \leq d \leq 120$$

Optimization problem is formulated as maximization of the objective function:

$$\max(\text{Obj_fun}(l_d, d)). \quad (27)$$

In Fig. 25 the Pareto front for optimization of a five-bar parallel micro robot for 4 optimization criteria, transmission quality index, workspace, manipulability and stiffness, is presented. For finding the Pareto front have been generated by a number of 500 generations. This approach focuses around the concept of Pareto optimality and the Pareto optimal set. Using these concepts of optimality of individuals evaluated under a multi objective problem, they each propose a fitness assignment to each individual in a current population during an evolutionary search based upon the concepts of dominance and non-dominance of Pareto optimality. More details regarding the developing the Pareto front can be found in (Stan, 2003).

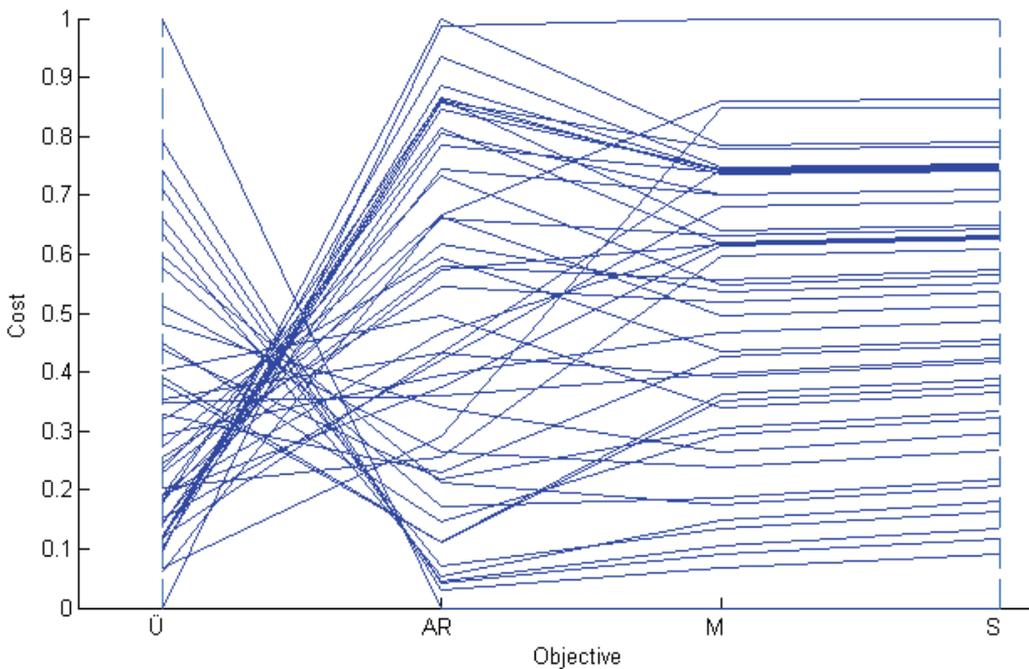


Figure 25. Pareto front for 4 optimization criteria: transmission quality index, workspace, manipulability and stiffness.

Since the finding of the solution for the multicriteria optimization doesn't end without choosing a compromise, there isn't need for an extreme precision for the values of the extreme positions. As Kirchner proved in (Kirchner and Neugebauer, 2000), optimization can be helped by a good starting population. The quality of the optimization depends essentially on the calculated number of generations.

In functioning of the genetic algorithms there have been used the following genetic algorithms parameters:

Size of population:	50
Generations:	10
Crossover rate:	0,07
Mutation rate:	0,01
Maximum number of generations:	500

Table 2. The GA Parameters

Based on the presented optimization methodology we can conclude that the optimum design and performance evaluation of the micro parallel robot is the key issue for an efficient use of micro parallel robots. This is a very complex task and in this paper was proposed a framework for the optimum design considering basic characteristics of workspace, singularities, isotropy.

4. Conclusion

An optimization design of 2-dof micro parallel robot is performed with reference to kinematic objective function. Optimum dimensions can be obtained by using the optimization method. Finally, a numerical example is carried out, and the simulation result shows that the optimization method is feasible. The main purpose of the chapter is to present kinematic analysis and to investigate the optimal dynamic design of 2-dof micro parallel robot by deriving its mathematical model. By means of these equations, optimal design for 2-dof micro parallel robot is taken by using GA. Optimal design is an important subject in designing a 2-dof micro parallel robot. Here, intended to show the advantages of using the GA, we applied it to a multicriteria optimization problem of a 2 DOF micro parallel robot. Genetic algorithms (GA) are so far generally the best and most robust kind of evolutionary algorithms. A GA has a number of advantages. It can quickly scan a vast solution set. Bad proposals do not affect the end solution negatively as they are simply discarded. The obtained results have shown that the use of GA in such kind of optimization problem enhances the quality of the optimization outcome, providing a better and more realistic support for the decision maker. Pareto front was found and non-dominated solutions on this front can be chosen by the decision-maker.

5. References

- Agrawal, S. K., (1990). Workspace boundaries of in-parallel manipulator systems. *Int. J. Robotics Automat.* 1990, 6(3) 281-290.
- Cecarelli, M., (1995). A synthesis algorithm for three-revolute manipulators by using an algebraic formulation of workspace boundary. *ASME J. Mech. Des.*; 117(2(A)): 298-302.

- Ceccarelli, M., G. Carbone, E. Ottaviano, (2005). An Optimization Problem Approach For Designing Both Serial And Parallel Manipulators, In: *Proc. of MUSME 2005, the International Symposium on Multibody Systems and Mechatronics*, Uberlandia, Brazil, 6-9 March 2005.
- Ceccarelli, M., (2004). *Fundamentals of Mechanics of Robotic Manipulation*, Dordrecht, Kluwer/Springer.
- Cleary, K. and Arai, T. (1991). A prototype parallel manipulator: Kinematics, construction, software, workspace results, and singularity analysis. In: *Proceedings of International Conference on Robotics and Automation*, pages 566-571, Sacramento, California, USA, April 1991.
- Davidson, J. K. and Hunt, K. H., (1987). Rigid body location and robot workspace: some alternative manipulator forms. *ASME Journal of Mech. Transmissions Automat Des*, 109(2); 224-232.
- Du Plessis L.J. and J.A. Snyman, (2001). A numerical method for the determination of dextrous workspaces of Gough-Stewart platforms. *Int. Journal for Numerical Methods in Engineering*, 52:345-369.
- Ferraresi, C., Montacchini, G. and M. Sorli, (1995). Workspace and dexterity evaluation of 6 d.o.f. spatial mechanisms, In: *Proceedings of the ninth World Congress on the theory of Machines and Mechanism*, pages 57-61, Milan, August 1995.
- Gogu, G., (2004), Structural synthesis of fully-isotropic translational parallel robots via theory of linear transformations, *European Journal of Mechanics, A/Solids*, vol. 23, pp. 1021-1039.
- Gosselin, C. (1990). Determination of the workspace of 6-d.o.f. parallel manipulators. *ASME Journal of Mechanical Design*, 112:331-336.
- Gosselin, C., and Angeles J. (1990). Singularities analysis of closed loop kinematic chains. *IEEE Trans Robotics Automat*; 6(3) 281-290.
- Gupta, K. C. (1986). On the nature of robot workspaces, *International Journal of Robotics Research*. 5(2): 112-121.
- Gupta, K. G. and Roth B., (1982). Design considerations for manipulator workspace. *ASME J. Mech. Des.*, 104(4), 704-711.
- Hesselbach, J., H. Kerle, M. Krefft, N. Plitea, (2004). The Assessment of Parallel Mechanical Structures for Machines Taking Account of their Operational Purposes. In: *Proceedings of the 11th World Congress in Mechanism and Machine Science-IFTOMM 11*, Tianjin, China, 2004.
- Holland, John H (1975), *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.
- Kirchner, J., and Neugebauer, R., (2000). How to Optimize Parallel Link Mechanisms – Proposal of a New Strategy. In: *Proceedings Year 2000 Parallel Kinematics Machines International Conference*, September 13-15, 2000, Ann Arbor, Mi. USA, [Orlandea, N. et al. (eds.)], pp. 307-315.
- Kumar, A. and Waldron, (1981). K.J. The workspace of mechanical manipulators. *ASME J. Mech. Des.*; 103:665-672.
- Masory, O. and Wang J. (1995). Workspace evaluation of Stewart platforms. *Advanced robotics*, 9(4):443-461.
- Merlet, J. P., (1995). Determination of the orientation workspace of parallel manipulators. *Journal of intelligent and robotic systems*, 13:143-160.

- Pernkopf, F. and Husty, M., (2005). Reachable Workspace and Manufacturing Errors of Stewart-Gough Manipulators, *Proc. of MUSME 2005, the Int. Sym. on Multibody Systems and Mechatronics Brazil*, p. 293-304.
- Schoenherr, J., (1998). Bemessen Bewerten und Optimieren von Parallelstrukturen, In: *Proc. 1st Chemnitzer Parallelstruktur Seminar*, Chemnitz, Germany, 85-96.
- Snyman, J. A., L.J. du Plessis, and J. Duffy. (2000). An optimization approach to the determination of the boundaries of manipulator workspaces. *Journal of Mechanical Design*, 122:447-455.
- Stan, S., (2003). Analyse und Optimierung der strukturellen Abmessungen von Werkzeugmaschinen mit Parallelstruktur, *Diplomarbeit, IWF-TU Braunschweig*, Germany.
- Stan, S., (2006). Workspace optimization of a two degree of freedom mini parallel robot, *IEEE-TTTC International Conference on Automation, Quality and Testing, Robotics - AQTR 2006 (THETA 15)*, May 25-28 2006, Cluj-Napoca, Romania, IEEE Catalog number: 06EX1370, ISBN: 1-4244-0360-X, pp. 278-283.
- Stan, S. and Lăpușan, C., (2006). Workspace analysis of a 2 dof mini parallel robot, *The 8th National Symposium with International Participation COMPUTER AIDED DESIGN - PRASIC'06*, Brașov, 9 - 10th November 2006, pag. 175-180, ISBN (10)973-653-824-0; (13)978-973-635-824-1.
- Stan, S., Vistriean M., Balan, R. (2007). Optimal Design of a 2 DOF Micro Parallel Robot Using Genetic Algorithms, *Proceedings of the 2007 IEEE-ICIT 2007, IEEE International Conference on Integration Technology*, March 20 - 24, 2007, Shenzhen, China, 1-4244-1092-4/07, p. 719-724, IEEE Catalog Number: 07EX1735, ISBN: 1-4244-1091-6, ISBN: 1-4244-1092-4.
- Stan, S., Balan, R., Vistriean M., (2007). Multi-objective Design Optimization of Mini Parallel Robots Using Genetic Algorithms, *IEEE-ISIE 2007 IEEE International Symposium on Industrial Electronics*, June 4-7, 2007, Caixanova - Vigo, Spain, IEEE Catalog Number: 07TH8928C, ISBN: 1-4244-0755-9, Library of Congress: 2006935487, pag. 1-4244-0755-9/07/ IEEE 2173-2178.
- Stan, S., Maties, V., Balan, R., (2007). Optimization of 2 DOF Micro Parallel Robots Using Genetic Algorithms, *IEEE-ICM 2007, IEEE - International Conference on Mechatronics 2007*, 8-10 May, 2007, Kumamoto, Japan, ISBN: 1-4244-1184-X IEEE Catalog Number of CD proceedings: 07EX1768C, ISBN of CD proceedings: 1-4244-1184-X, pp.1-6.
- Stan S., Maties, V., Balan R., (2007) Multicriteria Optimal Design of Two DOF Parallel Robots, *ISARC 2007- International Symposium on Automation & Robotics in Construction - 2007*, Kochi (Cochin), Kerala, India, 19-21 sept 2007, pag. 205-210.
- Sugimoto, K., Duffy J., Hunt K. H., (1982). Special configurations of spatial mechanisms and robot arms. *Mech Mach Theory* 1982, 117(2); 119-132.
- Tsai, Y. C. and Soni, A.H., (1981). Accessible region and synthesis of robot arm. *ASME - The Association for the Study of Medical Education Journal Mechanical Design*, no. 103, pag. 803-811.

Progressive Design through Staged Evolution

Ricardo A. Téllez and Cecilio Angulo
Technical University of Catalonia
Spain

1. Introduction

Evolutionary robotics is a good method for the generation of controllers for autonomous robots. However, up to date, evolutionary methods do not achieve the generation of behaviors for complex robots with a fixed body structure composed of lots of sensors and actuators. For such cases, no satisfactory results exist due to the large search space that the evolutionary algorithm has to face. Furthermore, the bootstrap problem does not allow convergence on the first generations, preventing the generation of simple solutions with a minimum fitness value that could guide the evolutionary path towards the final solution. Solutions like incremental evolution try to overcome the problem, but they do not scale well in complex robots with lots of devices.

The question is then, why natural evolution succeeded evolving complex animals, but evolutionary robotics does not. One answer to that question may be that, while natural evolution gradually evolved at the same time the animals body plan, their sensors and actuators, their nervous system, and even their environment, artificial evolution tries to evolve the nervous system for a robot with a fixed given body, sensors and actuators, within a fixed complex environment.

Our proposal states that when, as it happens in most cases, none of the evolutionary constraints can be relaxed (those are, the robot body, the sensors and actuators, the behavior to implement or the environment), then the use of external knowledge to guide the evolutionary process should be mandatory. Evolutionary approaches try to avoid the use of such knowledge, also called bias, because it directs the evolutionary search towards specific places of the space, not allowing the algorithm to find its own solutions. In this paper, we advocate instead for the use of bias as an inevitable situation when the robot body, task and environment are complex and fixed.

Based on this idea, we develop a modular architecture for evolutionary controllers based on neural networks, which allows the selective introduction of bias knowledge in the neural controller during the evolutionary process. The architecture allows the introduction of external knowledge on selected stages of the evolutionary process, affecting only selected parts of the controller that need to accommodate that information. The evolutionary controller is progressively designed in a series of stages, almost in a surgical way, independently of the complexity of the robot (in terms of number of sensors and actuators). This approach allows to avoid the bootstrap problem completely, and to obtain a completely distributed neural controller for the robot using only artificial evolution.

We will show how our method applies to general robots. Additional results will show its application to a complex Aibo robot.

The rest of the chapter is divided as follows: section two provides a description of the problem of evolutionary robotics that we try to solve, including a review of existing solutions. It follows a description of the architecture we developed to solve such problem, and a simple application to a simple wheeled robot. Then an application of the architecture to an Aibo robot, where more than 20 devices needed to be used to generate a behavior. The chapter ends with a discussion of the results, the conclusions and future lines of application.

2. Evolutionary robotics for complex robots

Evolutionary robotics is a framework for the automatic creation of autonomous robots. It reproduces on robots selective reproduction of the fittest, that means, it uses evolutionary algorithms to develop the control program of an autonomous robot (Nolfi & Floreano, 2000). The idea behind this framework is that the controller for a robot may be so complex that we don't know how to construct it. Instead we reproduce the mechanisms that evolution took for the generation of animal minds. The control system of the robot is then represented as an artificial chromosome subject to the laws of genetics and natural selection, which evolves on a hostile environment. Evolutionary robotics pays more attention to the processes that build a controller than how the final controller is. This liberates the designer of the necessity to specify how the controller must exactly work in every situation, and, more important than that, allows the evolutionary process to find novel solutions for the controller that the designer could not have thought of.

2.1 Complex robots

One of the most interesting problems of evolutionary robotics is the creation of behaviors for complex robots. We measure the complexity of a robot by its number of devices, that is, the number of sensors and the number of actuators that it has, which require a coordination in order to achieve a task, or generate a behavior. As this number increases, more difficult will be to generate an evolutionary controller for it.

At present, evolutionary robotics is mainly relegated to the realm of simple wheeled robots, where just a very small number of actuators (about two or three), and a small number of sensors (between two and six) are used. Even if quite complex behaviors have been evolved for such robots (Nolfi, 2004), there is no general solution for its implementation in complex robots with tens or hundreds of devices to be coordinated.

2.2 Problems faced when evolving complex robots

Two main problems arise in the evolution of controllers for complex robots: first, the search space that the evolutionary algorithm has to face to find a suitable solution is very large. Due to the large number of elements that need to be coordinated even for the simplest task, the number of weights of the neural controller is large. Then, the space of possible solutions has a high dimensionality, what makes computationally difficult to find a solution. The second problem is the bootstrap problem, which prevents the generation of simple solutions at the beginning of the evolutionary process with a minimum fitness value that could guide the evolutionary path towards the final solution (Nolfi & Floreano, 1998).

Up to date, there is no general method which avoids such problems, even if partial solutions have been proposed. For instance, incremental evolution (or incremental learning) is an evolutionary method that evolves a neural network by doing successive steps of teaching with an increase of the complexity of the task been teach on every step (Elman,1991; Gomez & Miikkulainen, 1996). This approach requires the use of the designer's knowledge of the task to correctly define the incremental evolutionary steps. It has worked well in complex tasks in simple robots, but a successful use in complex robots has not been reported. Modifications to this approach have also been proposed (Doncieux & Meyer, 2004), but their applications have still been applied to simple robots.

Co-evolution, is proposed as another solution. It consists of the evolution of competing controllers using a coupled fitness function (Nolfi & Floreano, 1998). However, co-evolution includes its own series of problems, and as (Doncieux & Meyer, 2004) say, they have to materialize yet.

In (Nelson et al. 2002), the authors propose a fitness function with two modes. The first mode rewards the controller when it has not been able to complete the task, using a subjective measure (completely determined by the designer) of the uncompleted task. A second mode only provides a reward when the task is achieved. A similar approach was also used by (Nolfi, 1997) on the generation of a garbage collector controller or on the generation of a hand able to grasp things (Bianco & Nolfi, 2004).

Another line of research involves the evolution at the same time of the robot body plan and the neural controller. Impressive results have been obtained in (Hornby and Pollack, 2002; Pollack et al., 2003). However, those approaches lack the ability to direct the evolutionary path of the body plan towards a body structure that has been designed before hand. But it is its lack of directness towards a given and fixed body what makes this approach useless for the case of a robot body which is fixed and complex.

We could however, introduce additional terms in the fitness function and restrict the possible morphologies to evolve a body plan similar to the one we want to control (the given robot body). This is the approach followed by (Muthuraman, 2005). Muthuraman performed a human directed evolution of a robot body and its neural controller in progressive steps. Initially, a simple controller for a simple version of the robot is evolved. When the controller cannot improve its fitness anymore, new neural modules are added to it, and the evolutionary process continued, until fitness improvement is not obtained. Then the body of the robot is complexified in some sense, and the previous evolutionary process repeated. The algorithm will continue complexifying body and controller until it reaches the robot body and task required. It seems difficult, though, to see how this method could be applied to any robot on any task, aside the one described by the author.

Additional solutions for the evolution of complex controllers include all sort of tricks for the particular controller to evolve. For example, (Mojon, 2004) generated the initial weights of the nets for a walking humanoid with small values, allowing the robot to obtain some controllers that didn't fall at the first generations. (Reeve, 1999) used symmetry on the evolution of a walking robot to reduce the search space and avoid initial useless controllers. (Ijspeert, 1998) used staged evolution for the generation of a controller for a salamander. In a similar way, (Lara et al., 2001) evolved two separated and independent neural controllers for a Khepera robot and, once they had those two modules, they evolved a neural interface between them, allowing the robot to show both behaviors on a unique controller.

3. Solution proposed

What all the previous approaches share in common is their introduction of human knowledge into the process in some or another way, that is, there is an introduction of bias by the designer. The introduction of bias helps to direct the evolutionary process towards a solution, reducing by hence the search space. However, this directness is usually not welcomed in evolutionary processes, since it performs a restriction on the solutions that the process will be able to find. The fact is that the bias leaves a short space for the algorithm to find a different solution from that that the researcher knows.

In our solution, we see the massive use of external knowledge as something inevitable in evolutionary robotics when the robot body, task and environment are fixed and complex. Starting from this idea, we propose a massive modular architecture for the control of robots, which allows the introduction of information in a selective manner, including certain information only on specific parts of the controller, without affecting others. The generation of the final controller is performed on a series of stages where the modules are progressively evolved using their specific information as fitness function, until the global controller is obtained. In fact, the global neural controller is progressively designed, through a series of evolutionary stages.

3.1 Description of the architecture

To generate a controller for a complex robot we think unavoidable the use of the concept of modularization. In neural controllers, modularization can be performed at two levels: at the structural level and at the learning level (Auda & Kamel, 1999). A modularization of the neural structure means that the controller will be composed of several neural networks. A modularization of the learning procedure implies that the learning will be performed in different stages. The problem is that it is difficult to train globally a chained functionality on a modular neural structure, because, even that we want a global intelligent behavior, the learning capabilities only exist at the organization level of the neural nets. Then, if modularization at the neural structure is taken, a modularization of learning may also be required.

Hence, for the progressive design of a neural controller we take modularization at both levels. We provide a standard building block and evolutionary method for any robot, independently of its number of sensors or actuators. The key point of this modularization is in which way the controller modularization is performed. While most of the existent modular architectures perform modularization at a functional level (that is, the global task to be implemented by the robot is decomposed into simpler subtasks (Davis, 1996; Lara et al. 2001; Dorigo & Colombetti, 2000)), in the progressive design method, modularity is created at the level of the robot device by creating a small and independent neural module around each of the sensors and actuators of the robot.

The evolution of the neural weights for each module is performed in several stages. On a first stage, only a small set of the sensors and actuators available are used for the evolution of a simple task, called an evaluation task, which must be related with the final global one. The neural modules associated to the set of sensors and actuators are evolved at the same time on the evaluation task. As the small group of controllers gains proficiency in their evaluation task, new sensors and actuators together with their neural modules are added to the group, and the evolution process started again with a new evaluation task. The re-started evolutionary process only affects the evolution of the weights of the newly added

modules and their connections with the evolved in the previous stages. As this process goes along, more and more sensors and actuators of the robot are progressively added, and the evaluation task they perform modified, until they reach the total number of sensors and actuators and the final global task required.

The evolution of modules by stages allows us to minimize the effects of the two main problems described in section 1: first, by evolving a limited number of modules at each evolutionary stage on a simplified task, we keep reduced the searching space the evolutionary algorithm has to face. Second, by evolving new modules keeping the functionality obtained in previous stages, the likelihood of obtaining a bootstrap is reduced, since the evolution of the newly added modules depart from a previous stable solution which can be scored, and by hence, directs the evolutionary path from the initial generations.

3.2 Sensor and actuator neural modules

Modularity is implemented at the level of the robot device and works as follows: each sensor and actuator will have an associated neural module, that we call in general, an Intelligent Hardware Unit (IHU). Each IHU module will belong only to that particular sensor or actuator. We will call those modules in particular as *SENEMO* (from Sensor NEural MOdule) for the union of a sensor and a neural net, and *ACNEMO* (from ACtuator NEural MOdule) for the union of an actuator and a neural net. Each neural network will *take care* of its associated device. This means that it decides which commands will be sent to its actuator, in the case of an ACNEMO, or how will the information coming from its sensor be processed, in the case of a SENEMO.

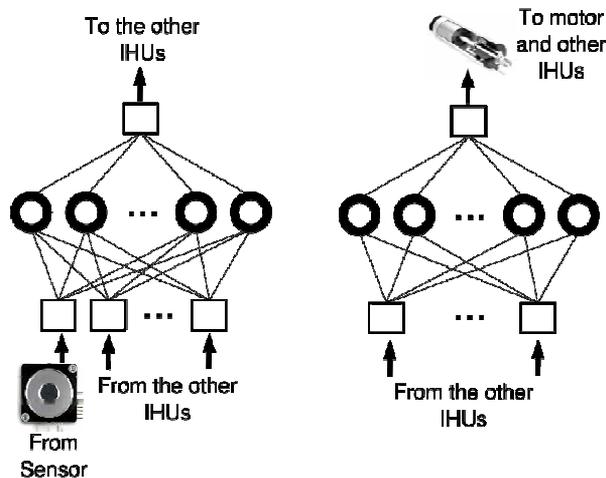


Figure 1. Left: Connection schematics of the neural modules for a sensor (SENEMO) and an actuator (ACNEMO)

The generation of a coordinated global robot behavior will require the cooperation between all the sensors and actuators. This is what is called sensorimotor coordination (Pfeifer & Scheier, 1997). In order to achieve such coordination, a communication channel is established that allows the modules the interchange of information. The communication channel is implemented by connecting the neural modules' output to one input of the other modules. This connection mechanism makes all the modules aware of what the other

modules are doing before deciding their own action. More complicated connection schemes are possible where two or more outputs were allocated in the modules, establishing one of the outputs as the action of the module, and the other as the communication for the other modules. We have kept this scheme simpler by allocating one single output for each module which acts at the same time as the output of the module and the communication for the other modules. Schematics of the SENEMO and ACNEMO module connections are included in figure 1. Figure 2 shows a connectivity example of the final controller obtained for a simple robot composed of two sensors and two actuators.

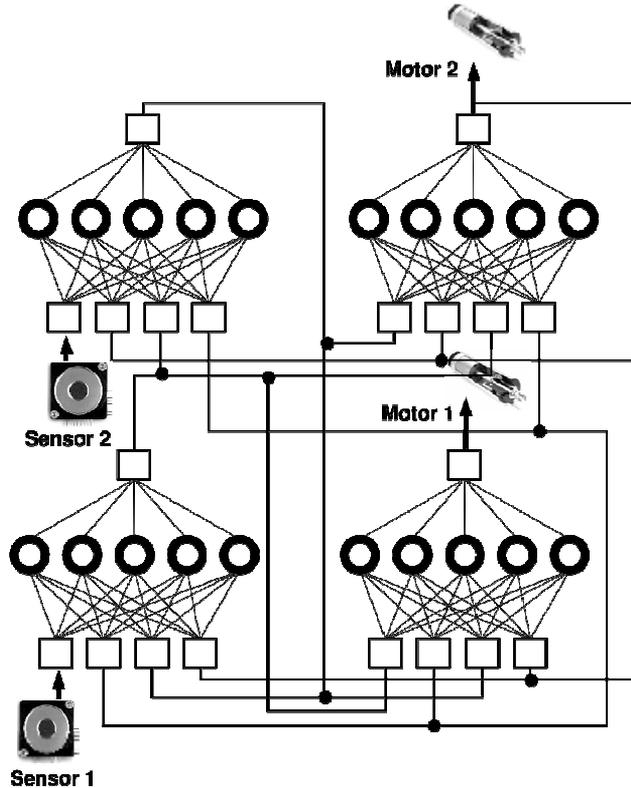


Figure 2. Application of the approach to the control of a simple robot composed of two sensors and two motors. Four modules are required, two SENEMO and two ACNEMO

It should be stated that when put several modules together on a control task, each module has its own particular vision of the situation since each one is in charge of its own sensor or actuator. Since there is no central coordinator, each module selects an action for its associated actuator or sensor output, based on the information sent by the other modules and its current status of its associated device, for the generation of the global robot behavior.

3.3 Evolutionary process

In the context presented above, the neural modules need to learn the control of their associated devices at the same time that they cooperate between each other for the generation of the global robot task. This is accomplished through an evolutionary process using a neuro-evolutionary algorithm. Due to the fact that the evolutionary process has to

evolve different neural networks for different roles on a common task, a co-evolutionary algorithm is required, that is, the simultaneous evolution of several nets, each one with its own population, but with a common fitness for all of them. By using such kind of algorithm it is possible to teach to the networks how they must cooperate to achieve a common goal (i.e. the global robot behavior to implement) when every network has its own an different vision of the whole system, since their inputs and/or outputs are different.

The algorithm selected to evolve the nets is the ESP (Enforced Sub-Populations)(Gómez & Miikkulainen, 1996), which has been proved to produce good results on co-evolutionary processes (Yong & Miikkulainen, 2001). A chromosome is generated for each module network, coding in a direct way the weights of the network connections, and the whole group of neural nets is evolved at the same time with direct interaction with the environment. The fitness function which guides the evolutionary process is created by the designer, depending on the problem that the robot has to solve.

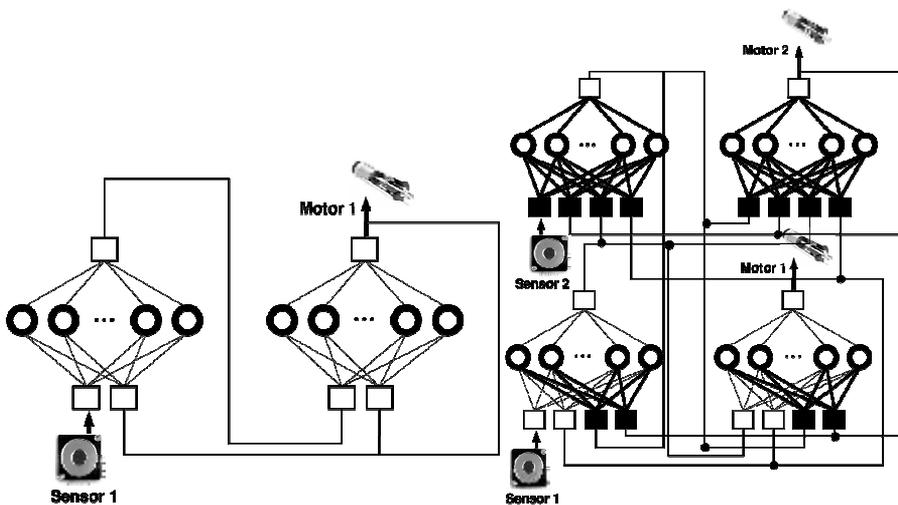


Figure 3. Progressive design of a neural controller for a simple robot with two sensors and two actuators. On stage one (left), only one SENEMO for sensor 1, and one ACNEMO for motor 1 are evolved obtaining the controller of the top of the figure. On stage two (right), another SENEMO (sensor 2) and ACNEMO (motor 2) are added to the controller. At that stage, only those newly added modules and their connections to the previously evolved modules are evolved (in thick black)

The progressive design method goes as follows: first, a goal-task for the robot is decided. This goal-task will be the behavior required for the robot. Then, the robot is divided into modules, one module for each sensor (a SENEMO) and actuator (an ACNEMO). Then the staged evolution begins. At the first stage, a subset of the modules are evolved to perform a reduced task related in some way with the goal-task. This reduced task is called an evaluation-task. The subset of modules to be evolved and the evaluation-task are selected by the designer, depending on the evolutionary strategy he is going to perform. The evaluation-task is usually a task that is relevant for the goal-task and that concerns the modules selected for this stage.

Once this evaluation-task has been achieved by the subset of modules with a reasonable fitness value, a new evolutionary stage starts where a new set of modules is added to the controller.

For this new stage, the evaluation-task may be changed, by incrementing its complexity, by changing it completely, or by not changing it at all. During this second stage, previously evolved modules are not evolved, and only the new ones added and their connections to the already existing ones are evolved on the new evaluation-task. Once the evolutionary process obtains a reasonable fitness value, a new stage begins. This procedure is repeated until all the modules are included in the controller and the final goal-task is achieved.

Figure 3 shows the two stages progressive design of a controller for a robot composed of two sensors and two actuators.

3.4 Application example

In this section we will show an application example of the methodology to a simple robot, in order to clearly see how it works. The example will be the Khepera garbage collector. An application to a complex robot in a complex task will follow in the next section.

Experiments consisted of the implementation of the progressive design method for the control of a Khepera robot while performing a cleaning task. The selected test bed task is called the garbage collector, and follows the following description:

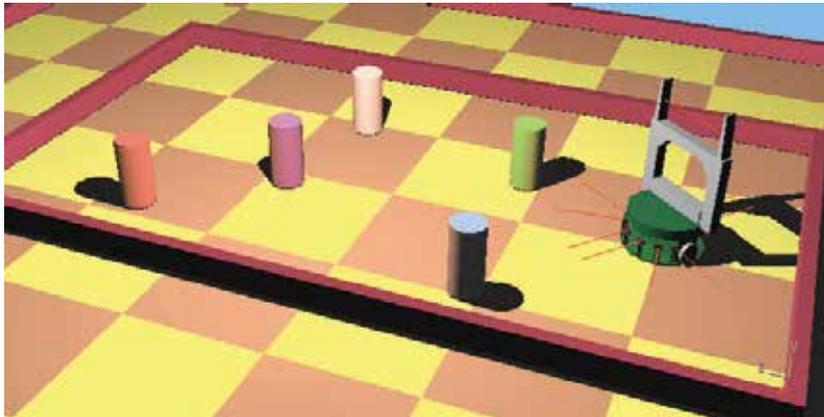


Figure 4. The garbage collector experiment on Webots simulation environment

A Khepera robot is placed inside an arena of dimensions 60x35 cm surrounded by walls (figure 4), where it must look for any of the sticks randomly distributed in the space, grasp it, and take it out of the arena. The robot was also placed randomly in the arena at the beginning of each epoch.

The robot has eight infrared (IR) sensors, six at the front of the robot, two at the back. It also has one presence detector sensor in the gripper fingers. For the task of the garbage collector, only the six front sensors will be used, as well as the gripper sensor. IR sensors have a limited range of detection (from 0 to 75 mm). The sensors have 1024 different values from 0 to 1023, which indicate the presence of an object at the corresponding distance. Objects placed at a distance above 75 mm are not detected at all.

As actuators, the robot has two motors (left and right), but it is also possible to control the position of the gripper arm and the status of the gripper fingers (open or closed). The gripper turret is controlled by two procedures that activate a complete behavior. When procedures are activated, they perform a complete series of movements of the gripper which cannot be interrupted until finished. In the case of the first procedure, the movements are:

move the arm down, close the gripper fingers. If there is nothing between the fingers, then open the fingers. Then move the arm up again. This procedure should be the one activated, in order to pick up a stick, and by hence we will call it the pick-up procedure. The second procedure moves the arm down, opens the gripper fingers, and moves the arm up again. It should be activated when a stick has to be released, so we will call it the release procedure. Of course, the controller does nothing neither about the behavior of those procedures nor when they have to be activated. The evolutionary process will have to create the appropriate controller that activates those procedures when it is necessary. Procedures are activated on a on-off basis, i.e., they will be taken as if they were actuators which can be controlled with two possible values: an on value when require activation, or an off value when no activation is required.

Experiment setup

Basically, experiments consist of 15 epochs of 200 time steps each, where an evolved controller was tested over the task. The duration of each time step was of 100 ms. Each epoch ended after the 200 steps or after a stick had been correctly released out of the arena.

The final goal is the generation of a modular controller for the Khepera robot solving the garbage collector. This is a controller composed of eleven modules: six infra-red sensors, one gripper sensor, two motors, and two gripper procedures. Each neural module was implemented using a feed-forward neural net with no hidden units and only one output. At the beginning, the number of inputs of those nets will depend on the number of elements selected for the first evolutionary stage. In any case, at the end of the progressive design, the neural net of each module will have eleven inputs. Figure 6-left shows the model of neural network used for each module. Figure 6-right shows the final controller that we want to obtain by progressive design.

The architecture is evolved using the progressive design process described above. A three stages procedure is designed. The evolutionary strategy decided by us consisted on using a reduced set of sensors to perform the garbage collector task. This means that, on a first stage, not all the sensors will be used in order to reduce the number of weights to be evolved. Then, in progressive stages, new sensors will be added that may allow increase the proficiency of the robot for the task, and by hence the fitness obtained. We decided to divide the progressive design of the controller into three evolutionary stages. On the first one, only seven modules are evolved (three sensors and four actuators are used). On the second stage, two additional sensors are included. On the final stage, the remaining two sensors are added.

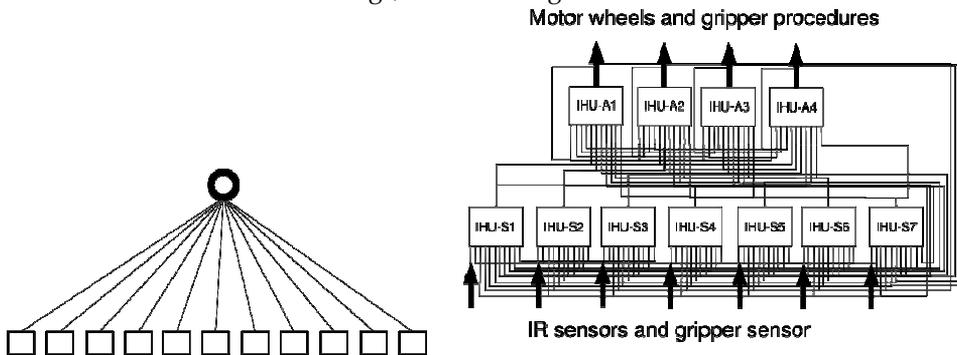


Figure 6. Left: The neural network used for each module at the end of the evolutionary process. Right: Modular representation of the architecture implemented for the Khepera robot

First evolutionary stage

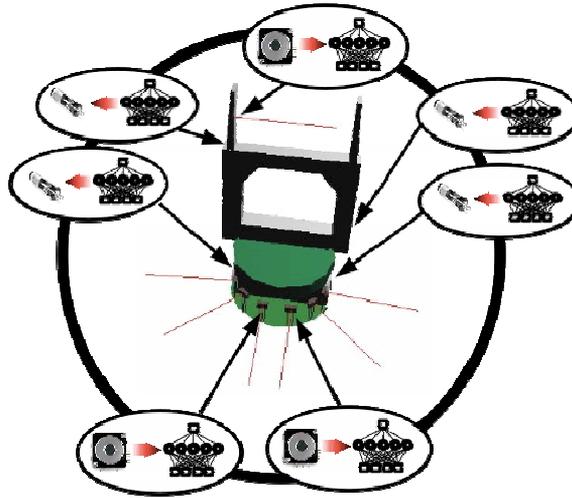


Figure 7. First stage evolutionary controller. At this stage, only seven modules are evolved corresponding to three sensors and four actuators

On the first stage, we start using as less sensors and actuators as possible, in order to have an initial search space small. However, we want the robot be capable of performing more or less the garbage collector task. So we decide that using the two front IR sensors would be the minimum necessary for the robot that will allow it to discriminate between sticks and walls. For grasping and releasing sticks the gripper sensor is required, as well as the two wheel motors and the take and release procedures. This makes a total number of seven IHUs (figure 7).

A fitness function was created for the evolutionary process which rewarded controllers capable of releasing one stick out of the arena. An additional term was added to the function which rewarded robust controllers, those are, controllers that performed the stick releasing behavior without performing any error. Errors included, crashing into walls, releasing sticks inside the arena or over another stick, or trying to pick a wall. Controllers that were able to only pick up one stick were also rewarded with a lower fitness.

$$\text{fitness} = \begin{array}{ll} 1 & \text{if pick up stick} \\ 100 & \text{if stick released outside arena} \\ 110 & \text{if stick released without errors} \\ 0 & \text{otherwise} \end{array}$$

After 1000 generations, the maximum fitness obtained was of 1531 (out of 1650). The behavior obtained, even that a little bit non-optimal, allowed the robot to solve the task. Due to the randomness of the method employed, we performed the evolutionary process ten times, obtaining a mean fitness value of 1021. Eight out of ten evolutionary processes were able to generate the garbage collector behavior. Figure 10 shows the evolution of the mean fitness value over generations.

Second evolutionary stage

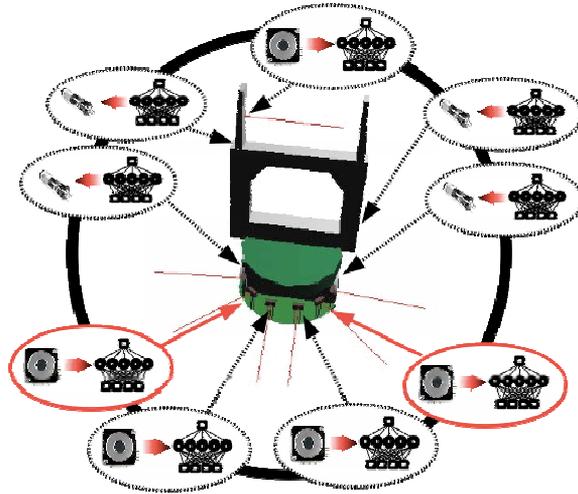


Figure 8. At the second stage, two additional sensor modules were added (in red)

On the second stage, the best group of modules evolved on the previous stage were frozen in their evolution. Two new sensor IHUs were added to the controller, the ones corresponding to the control of the two diagonal IR sensors (figure 8, in red). During this stage, only the weights of those two new modules and their connections to the already existing modules were evolved. New connections between modules are random initialized by the evolutionary algorithm, and evolved in separated sub-populations.

For this task, the fitness function used at this stage is the same as in the previous one. If the progressive design method were used in the evolution of other controllers, the fitness function may have changed at this point if necessary, in order to implement a different task starting with the knowledge of the task learned in the previous stage. For the garbage collector problem, it is not the case, and the same fitness function can be used, even if the structure of the controller has changed. An example of task that requires the change of the fitness function at different stages will be discussed in section 4.

Figure shows the evolution of the mean fitness value over generations for this stage. If we look at the evolution of fitness, we observe that, at the beginning of this new evolutionary stage, the fitness obtained at the first generation is lower than the maximum obtained at the previous stage. This is due to the interference that the newly added connections are producing in the solution found at the previous stage. However, the fitness of this stage first generation is not low, since the neural weights obtained in the previous stage represent a point in the fitness landscape where a solution is near, that is, the task learned in the previous stage represents a good starting point for the learning of this stage task. After 1000 generations the maximum fitness reached is 1650. We performed the evolutionary process ten times, obtaining a mean fitness value after 1000 generations of 1570. All ten evolutionary processes were able to generate the garbage collector behavior.

Third evolutionary stage

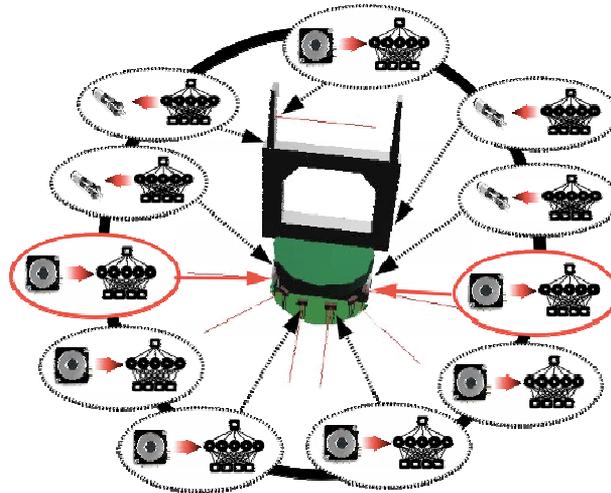


Figure 9: At the final stage, two additional sensors were included (in red)

At the third stage, the IHU for the two lateral sensors are introduced (figure 9, in red). The same procedure as in the previous stage is used. The already existing modules are frozen, and only the two new modules and their connections to the already existing ones are evolved. The same fitness function was used for this stage.

For this stage, we observe the same behavior as in the second stage, that is, the fitness obtained in the first generation of this stage is a slightly reduced value from the maximum obtained at the end of the previous stage. However, at this stage, the reduction is smaller than the reduction experienced at stage 2. Our hypothesis is that the solution found at the previous stage is very related to the solution with two more sensors. In fact, the task is the same but using two more sensors, so the solution has only to accommodate the action of the newly added sensors.

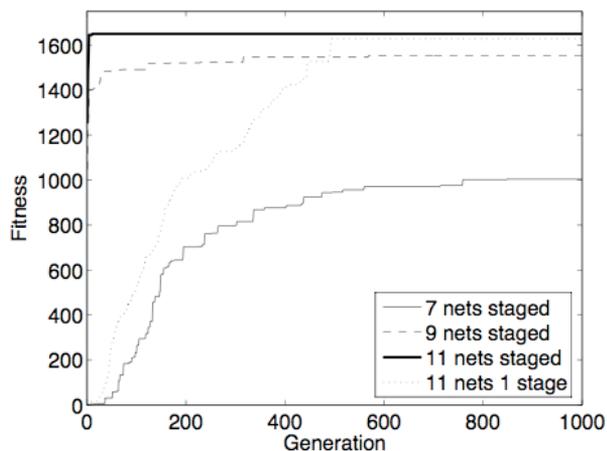


Figure 10. Mean fitness evolution through generations for all the stages. Dotted line shows the fitness evolution obtained when the eleven modules are evolved in one single stage. Values have been averaged over ten evolutionary runs in all cases

After 1000 generations the maximum fitness value is 1650. We performed the evolutionary process ten times, obtaining a mean fitness value after 1000 generations of 1647. All ten evolutionary processes were able to generate the garbage collector behavior. Figure 10 shows the evolution of the mean fitness value over generations.

Comparison with one single stage evolution

Due to the simplicity of the robot used, the simplicity of the task to be solved, and the power of the evolutionary algorithm, it is possible to evolve the whole controller of figure 6 for the garbage collector in one single stage. This implies to evolve at the same time, in one single stage, the eleven modules.

If we compare the case of one single stage evolution with progressive design, we observe that the mean fitness value of the former is 1645, a fitness slightly below the mean fitness obtained by progressive design (1647). Figure 10 shows the evolution of the mean fitness value over generations, compared with the evolution of fitness of the other stages. Furthermore, in the case of single stage evolution, nine out of ten evolutionary processes were able to evolve a garbage collector behavior, meanwhile in the case of the progressive design, all the controllers of the last stage were able to generate the garbage collector behavior (10 out of 10). Even if both approaches obtained similar results, a small improvement seems to be on the side of the progressive design. This results seems reasonable, since in the progressive case, the controller is carefully built step by step, starting from previously known domains, which allows for a better evolutionary process.

However, the improvement in fitness obtained it may not be worth it because of the needed complexification of the evolutionary process. This complex procedure may only be required in more complex situations where the single stage evolution is not possible. This is the case of the next section.

4. Complex robot application

The results presented in the previous section show how the progressive design method works for a simple case. However, the main aim of this methodology is its use in complex robots. This section shows the results we obtained when we applied the methodology to the Aibo robot for the generation of a walking pattern. The generation of such behavior is a complex issue, since Aibo has 12 degrees of freedom (DOF) related to walking (no head or queue DOF have been taken into account). Each leg has 3 DOF that must be coordinated with the other legs in order to obtain a walking gait. It is easy to see that an error of coordination in one single joint may cause the robot not walking at all.

For the generation of the walking gait we will only take into account the 12 motors of the leg joints as well as those joints sensors, what makes a total of 24 devices which need to be controlled. The whole process is carried in simulator and once the global controller is obtained, it is transferred to the real robot.

The generation of a walking gait for a quadruped has been solved in the literature using different methods. In order to feed our evolutionary controller with the required information, we studied the different solutions already available. One of the most interesting was the use of non-linear dynamic central pattern generators (CPGs) (Collins & Richmond, 1994; Lewis, 2002). The idea is to use the information about how the robot should walk to feed the evolutionary process.

4.1 The Central Pattern Generator concept for walking robots

It has been shown that animals perform rhythmic movements such as walking or running by means of CPGs (Grillner, 1985). CPGs are groups of neurons that generate an oscillatory pattern from a tonic input signal. Depending on the value of the tonic signal, CPGs can change its frequency of oscillations as well as its amplitude.

CPGs can be reproduced artificially using artificial neural networks. Those CPGs are the ones that will be created in the robot by using the distributed architecture. In real animals, each joint has a CPG for its control and these are interconnected only with the nearest CPGs. In the robot case, Aibo's joints are composed of a sensor (which obtains the position of the joint at each instant) and an actuator (which moves the joint). In our architecture a CPG is implemented for each joint by coupling a neural net for the joint sensor and a neural net for the joint actuator (see figure 13). The final robot controller can be seen then as a group of CPGs coupled ones with the others.

Because such a controller cannot be evolved in one single round by any current evolutionary algorithm, a progressive design of it is proposed. The different stages for the generation of the walking gait are: generation of the CPG oscillator, where a segmental oscillator is evolved for each type of joint; generation of a layer of joints of the same type that oscillate in counter phase by using the previously generated CPGs; and coupling of the three layers to obtain the final walking behavior.

4.2 Neural model used

We begin by analyzing the type of neural network that we will use in the IHU models for the generation of the gait controller. Due the dynamic nature of the task to evolve, a neural network capable of capturing dynamics is required. We find in (Reeve & Hallam, 2005) an exhaustive analysis of the characteristics, advantages and drawbacks of different types of neural networks for walking behaviors. We select the model called Continuous Time Recurrent Neural Nets (CTRNN).

CTRNNs are composed of a set of neurons modeled as leaky integrators that compute the average firing frequency of the neuron.

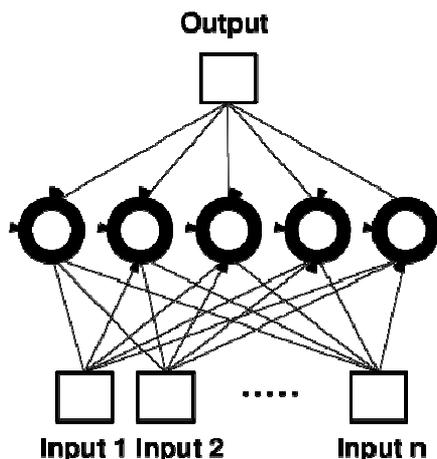


Figure 11. Schematics of the CTRNN used in the walking controller. The hidden units are modeled as leaky integrators

The output of each hidden unit of a CTRNN network is computed using the following equations (from (Reeve & Hallam, 2005)):

$$\tau_i \frac{dm_i}{dt} = -m_i + \sum w_{ij} x_j \quad (1)$$

$$x_i = (1 + e^{(m_i + \theta_i)})^{-1}$$

where m_i represents the mean membrane potential of neuron i , that is, the output of the network. x_i is the short-term average firing frequency of neuron i , θ_i is the neuron bias, τ_i is a time constant associated with the passive properties of the neuron's membrane, and w_{ij} is the connection weight from neuron j to neuron i . Calculation of each neuron output is performed using the Euler method for solving differential equations with a given step of 96 ms.

A neural net similar to the one shown in the figure 11 is used for each IHU neural element, where: the number of inputs equals the number of devices to control, i.e. 24; the number of hidden units is five; and the number of output units is one. Inputs represent the connections of a given IHU to the others, and the output is the answer from that IHU to its associated element (sensor or actuator).

For each hidden unit of the networks it is necessary to evolve its connection weights to inputs and outputs, its bias term, and its time constant. So, the genotype for each hidden unit will contain those values in a direct encoding scheme. Initially, the values of the parameters for all the neurons are randomly created around certain values. Weights are initialized between -16 and 16. However, for the bias term and the time constant different initializations were done based on the data of (Seys & Beer, 2004). For the bias, the initial values were taken between -16 and 16. For the time constant, values were between 0.5 y 10.

4.3 Progressive design of a walking gait

The evolutionary strategy for the evolution of the controller makes intensive use of the CPG concept for the generation of the controller. Using that information, we decide that the controller of the robot will be made of a group of CPGs, one per each joint, that will drive its corresponding joint with an oscillation signal. Joint oscillators will be coordinated in the appropriate way to produce a walking gait.

Once those points were clear, its practical implementation was divided into four stages: on a first stage, a controller that performs an oscillation is generated for each joint using its associate motor an sensor. At that point each joint has its own an independent oscillator that drives it. On a second stage, two oscillators of the same joints in different legs are coupled in the form that they generate a single controller that drives both joints with an oscillation, but with a required phase shift between them. On the third stage, two groups of two coupled oscillators are again coupled, obtaining a group of four oscillatory joints driven by one single controller. Those three stages are repeated for the three types of joints that the robot has, what leads to a situation where three controllers exist, each one driving four joints on a synchronized oscillation. The fourth stage evolves the connections between those three controllers, generating a single controller for the whole robot, which finally is able to walk.

First stage: generation of the joint oscillator

The aim of this stage is to obtain a controller for a joint of the robot, capable of generating an oscillatory pattern for each type of robot joint. Joints in the robot's legs are of three different types, which we will call J1, J2 and J3 (see figure 12). J1 is in charge of the rotatory movement of the shoulder, J2 of the lateral movement of the shoulder and J3 of the knee movement. Each joint is physically implemented using different PID controllers, and, in addition, their movement limits are different. For this reason, a different type of oscillator must be implemented for each type of joint, this means, we are going to evolve three types of oscillators. The evolution of the oscillator for each joint type is performed separately. It means that each joint type will evolve its associated controller on a different and independent evolutionary process. Nevertheless, the process for the generation of each type is exactly the same, with the only practical difference being the range of movements, and the way the leg is moved. So, at the end of this stage we will obtain three different controllers, one for each joint type.

For each joint, an oscillator is implemented by coupling two CTRNN networks: one for the joint sensor and another for the joint actuator (the motor). The resulting controller is shown in figure 13. Both nets are interconnected as the architecture specifies, but each one is in charge of a different element (the sensor net is in charge of the sensor, and the motor net is in charge of the motor). At each time step of the evaluation process, the sensor reading is taken and entered into the IHU sensor. Then the output is computed and sent to the IHU actuator. The output of the IHU actuator specifies the velocity to be applied to the motor, and, once de-normalized, it is applied directly to the motor.

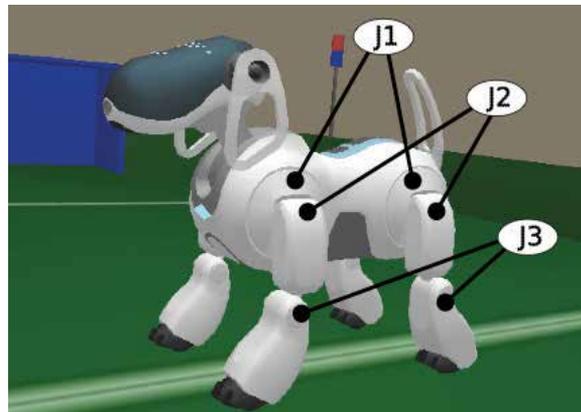


Figure 12: Detail of the three different joint types for the robot legs

The weights of the nets are evolved using the ESP algorithm and a fitness function that rewards the production of an oscillatory pattern in the motor joint. The type of pattern that should be obtained is not specified, only that it has to be periodic and between certain oscillatory limits. The following fitness function was defined for the evolution of such oscillations:

$$mean = \frac{1}{N} \sum x_i$$

$$fitness = \frac{1}{N} \sum (x_i - mean)^2 \quad (2)$$

Basically, what this fitness function calculates is the variance of the trajectory followed by the joint. Thus, the evolutionary process will try to maximize it, and the value of the variance will be maximum when the position of the joint changes from one limit to the other.

By using the previous fitness function, we obtained a single oscillation within the full range of the joint, that is, the joint oscillated one single time from one limit to the other. Aibo joints can oscillate between very large limits, but those are too large for an appropriate walking behavior, and we needed to limit them, by defining some oscillation limitations. In order to keep it simple we took those limits by observing the walking limits of the walking style that comes with the Aibo robot from Sony. Those limits are included in table 1.

A new fitness function was defined to reward regular oscillations within the limits of each joint. In order to obtain this, it is required that the system should generate a joint movement around the mean value in the table, with maximal variance within the limits for each joint. The fitness function is then a product of two factors:

$$fitness = fit_var * fit_cross \quad (3)$$

where *fit_var* is the variance of the position of the joint during the 200 evaluation steps, as calculated in equation (2). *fit_cross* is the number of crossings that the joint performs through the mean position value. The addition of the second term to the fitness function ensured that the final oscillation obtained would not be limited to a single oscillation. However, this term also ended in the generation of very high frequency oscillations, which increased the second term of the fitness function. In order to avoid that, a limitation in the number of possible crossings was established to 20, what means that bigger number of oscillations were taken as 20.

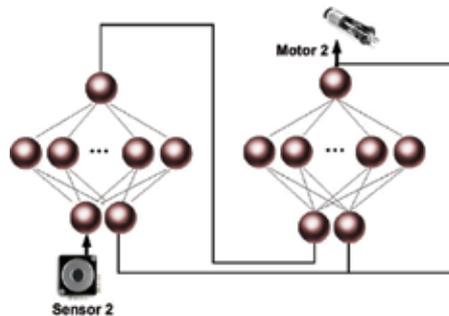


Figure 13: Schematics of the coupling between two neural nets of a joint. The figure shows the coupling between the joint sensor IHU (SENEMO) and the joint motor IHU (ACNEMO)

Joint	Max	Min	Mean
J1, fore	0.3936	-0.5837	-0.0950
J2, fore	0.3702	-0.2163	0.0769
J3, fore	1.1732	0.1435	0.6583
J1, rear	0.0059	-0.7848	-0.3894
J2, rear	0.4215	-0.2163	0.1026
J3, rear	1.6599	0.9907	1.3253

Table 1: Limits taken for Aibo joints oscillations. It includes a calculated mean value of the range of each joint. This mean value establishes the central position of the joint around which the joint will have to oscillate. Values are in radians

On addition, each fitness factor was limited by a function that linearly varies between 0.01 and 1.0 when its corresponding variable fluctuates between a good and a bad boundary (see table 1). This means that normalization was applied to each factor in order to have a fitness value between 0.01 and 1 for bad or good results. This transformation allowed a clear vision of the evolutionary state at any time.

For the results, ten runs were carried out for each type of joint starting with different initial random populations. Each run was composed of 200 simulation steps of 96 ms. After 13 generations all runs converged to networks capable of maintaining oscillations within the range specified.



Figure 14: Sequence of the oscillatory movement obtained in simulation for the J1 joint type. The robot is lying on his side to allow the movement of the joint freely. The evolution of the other joint types was performed with the same setup

Second stage: generation of two coupled oscillators

In the previous stage, three different and independent controllers were obtained, each one for the control of one type of joint of the robot. The aim of the second stage is to obtain a controller for two joints of the same type, which makes them oscillate with a given phase relationship.

For this case, if we apply the architecture, this means to have a controller composed of four neural networks (for each type of joint): two for controlling the two motors and two for controlling the two sensors. Since we evolved in the previous stage the controller for one joint and its sensors, in this case it will only be required to evolve the neural modules for the newly added motor and sensor, as well as the connections between those newly added modules and the already existing for the previous joint. However, a quicker and simpler solution exists, and it consists of performing a duplication of the modules created in the previous stage for the new joint. This means that the oscillator that controls one type of joint of the previous stage, is going to be copied to control the joint of the same type that is in the opposite side to the original one.

Then connections between both modules are established in order to apply the architecture definition. Since the interior of the copied networks were already evolved in stage one, only the connections between modules are going to be evolved in this stage (see figure 15). It is important to note that at this stage, the three types of oscillators are still completely independent yet from each type and evolved in different processes. So each evolutionary process has to cope with a reduced search space only related to its associated joint type. Also note, that it could also be possible to formally apply the tactical modular method not copying the modules and evolving completely the new joint controllers, but in this case we could take advantage of this situation to reduce the search and the evolutionary time.

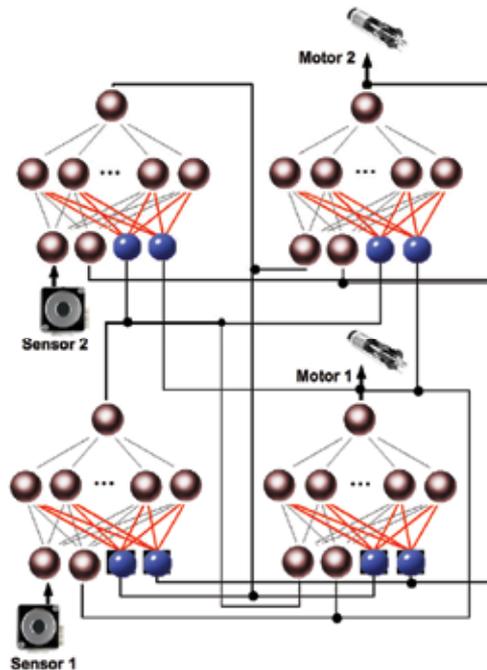


Figure 15: Connections between four IHUs corresponding to two joints of the same type

Therefore, for each type of joint, the controller obtained in the previous stage is copied from the left leg joint to the right leg joint, obtaining by hence two isolated oscillators like the one of figure 13, each one controlling their associated joint. However, the controller to be created requires that all the joints oscillate with a determined phase relationship. This phase relationship is determined by the type of gait that one wants to implement. The coordination between joints is achieved by creating the connections between both modules and evolving them. Figure 15 shows the final controller for the control of two joints in counter phase. Weights in red are the ones to be evolved.

Leg	Walk	Trot	Bound
Leg-Fore	0°	0°	0°
Right-Fore	180°	180°	0°
Left-Rear	270°	180°	180°
Right-Rear	90°	0°	180°

Table 2: Phase relationship for three common quadrupedal gaits (Collins & Richmon, 1994)

Then, connections are made between the two groups of nets. This implies that each neural net will have to add two more inputs coming from the outputs of the other two neural nets duplicated. Once in the evolutionary process, only the new connections between IHUs will be evolved: neither the internal connections of the neurons nor the time constant and bias obtained from the previous stage will be evolved. Since the oscillation has already been obtained in the previous stage, it will not have to be evolved in this stage, although the phase relationship between the two oscillators will be required.

In this case, we are interested in the implementation of a walking gait, which implies a phase relation of 180° between those two legs (in all types of joints). The fitness function will be that which punctuates the phase difference between the legs close to 180°, and rewards a continuous oscillatory movement of both legs. The fitness function is composed of three parts: two parts are the fitness function of the first stage for each leg; the third part is the fitness factor that measures the variance between the movements of both legs, and tries to maximize it.

$$fitness = fit_cross_1 * fit_cross_2 * fit_var \quad (4)$$

where fit_var is the variance of the difference of positions between both legs during the 400 evaluation steps, and fit_cross_1 and fit_cross_2 are the number of crossings that each joint performed through their mean position value. Similar to the fitness function for the first

stage, these factors vary between 0.01 and 1.0 when their corresponding variable fluctuates between a good and a bad boundary.

For the results: ten runs were carried out for each type of joint to evolve the connections between CPGs. Each run was composed of 400 simulation steps of 96 ms. After 14 generations, 90% of the networks were capable of a counter-phase oscillatory pattern.



Figure 16. Sequence of the oscillation obtained in simulation for the J1 joint type. The robot is sit on its butt to allow a free movement of the joint. The evolution of the other joint types was performed with the same setup and similar behaviors were observed

Third stage: coupling the oscillation of four joints of the same type

In this stage, the same procedure as in stage two was implemented. Now the control for the rear two joints of the same type is added to the controller. This means that four new neural modules will be added to the modular controller, two for the control of the two rear joints and two for the two rear sensors. However, since for a walking gait the two front joints have the same phase relationship as the two rear joints have, it is possible to clone the controller obtained for the front joints to the rear joints, and then evolve the connections between them. Since four new modules are added for the control of the two rear joints, it is necessary to evolve 4 connections per network, with a total number of 8 IHUs per joint type.

The evolution of the new connections is performed using a fitness function which rewards controllers that achieve an oscillation of all the legs with a given phase relationship, as specified in table 2. For a typical walking gait the relation from left to right and from front to rear is $0^\circ, 180^\circ, 270^\circ, 90^\circ$.

Hence, making use of that algorithm, the fitness function designed to obtain coordination was composed of three parts: a part for each leg that expresses the oscillation requirement; a part that expresses the maximal variance requirement between the fore legs; and a final part that expresses the maximal variance requirement between fore-rear differences.

This is specified in the following fitness function:

$$fitness = fit_cross * fit_oscil * fit_phases \quad (5)$$

where fit_cross is the product of the number of crossings for each joint performed through their mean position value, fit_oscil is the variance of the left fore joint which indicates how well that joint oscillates (if this joint oscillates, the others must follow), and fit_phases is the part of the fitness that indicates the phase relationship between all the joints. Similar to the fitness function for the previous stage, these factors vary between 0.01 and 1.0 when their corresponding variable oscillates between a good and a bad boundary.

For the results: the evolutionary process was carried out ten times. Each time, it was composed of 400 simulation steps of 96 ms each. After 26 generations, 92% of the networks were capable of the typical oscillatory walking pattern $0^\circ, 180^\circ, 90^\circ, 270^\circ$ (for the legs sequence fore_left, fore_right, rear_left, rear_right).

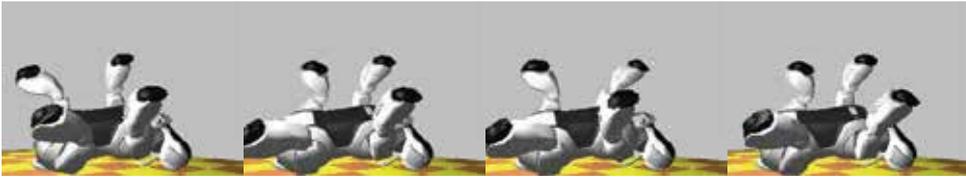


Figure 17: Sequence of the oscillation obtained in simulation for the J1 joint type. The robot is lying on its back to allow a free movement of the joint. The evolution of the other joint types was performed with the same setup and similar behaviors were observed

Fourth stage: coupling between types of joints

The last stage is the coupling between the three groups of neural controllers obtained. From the previous stage three different oscillating modular controllers were obtained, one per joint type, with four joints of the same type oscillating together with a walking phase relationship. It is now required to interconnect the three layers in order to obtain a coordination between the different joint types, that enables the robot to walk, and completes the architecture as a whole. The next step will be the evolution of the connections between the three groups of controllers. In terms of walking, connection between groups should produce coordination between the different types of joints that have been evolved separately.

The connection between the three groups of controllers implies that 16 new inputs will be added to each IHU neural module. Those inputs represent the connection to the other 16 modules of the other two groups. Only those connections between groups are evolved to generate the required coordination between the groups for the generation of a stable walking.

On a first approach, we tried to evolve the coordination between groups with a simple fitness function composed of the distance walked by the robot. However, the walking behavior obtained by that approach, even if correct, was very sudden and induced instabilities that made sometimes the robot fall. Analyzing the behavior obtained, we observed that the coordination between groups was correctly achieved but some of the joints had loose their oscillation pattern.

Because of that, a new fitness function was proposed where the oscillation of the joints was still imposed, together with the distance walked. If the robot does not fall over, the fitness function is composed of two multiplying factors: the distance d walked by the robot in a straight line and the phase relationship between the different joints. In the event of the robot falling over, the fitness is zero.

$$fitness = \begin{cases} d * fit_phases & \text{when final height} > 0 \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

For the results: a walking behavior was obtained after 37 generations for around 87% of the populations. A walking sequence obtained is shown in figure 18.

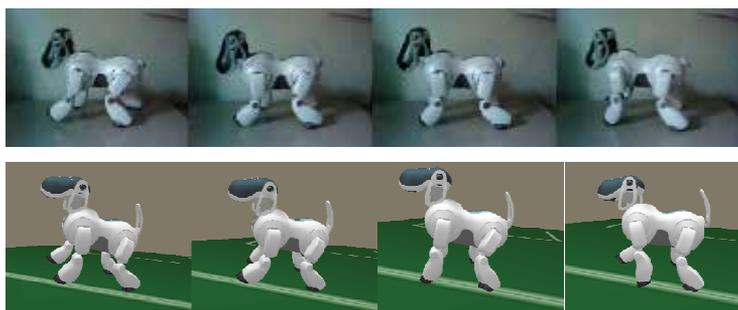


Figure 18: Top: Real Aibo walking sequence. Bottom: Simulated Aibo walking sequence

Once this walking behavior was obtained in the simulator, the resulting ANN based controller was then transferred to the real robot using the Webots simulator cross-compilation feature. The result was an Aibo robot that walks in the same manner as the simulated robot with some minor differences. A walking sequence obtained is shown in figure 18.

5. Discussion

The progressive design method allows for the evolution of complex controllers in complex robots. However, the process is not performed in a complete automatic way as the evolutionary robotics approach aims to have. Instead, a gradual shaping of the controller is performed, where there is a human trainer who directs the learning process, by presenting increasingly complex learning tasks, and deciding the best combination of modules over time, until the final complex goal is reached. This process of human shaping seems unavoidable to us if a complex robot body, sensors/actuators, environment and task are imposed before hand. This point has also been suggested by other researchers (Urzelai et al. 1998; Muthuraman et al., 2003). But, different to other approaches, progressive design, by implementing modularity at the level of devices and also at the level of learning, allows for a better flexibility in terms of shaping of complex robots. The main reason is that, due to the modularization at the level of device, the designer can select at any evolutionary stage which small group of sensors and actuators will participate, under which task, and only evolve those modules. This would not be possible on a complex robot if modularization at the level of behavior is used.

Progressive design can be seen as an implementation of the incremental evolution technique but with a better control of who is learning what, at each stage of the evolutionary process. If incremental evolution were used on a controller with several inputs and outputs which controls every aspect of the robot, it would be possible to produce genetic linkage by which, learning some behaviors on early stages would prevent learning other behaviors in following steps, because the controller is so biased that it cannot recover from. This effect may be specially important in complex robots where several motors have to be coordinated. The learning of one coordination task may prevent the learning of another different one. Instead, the use of progressive design allows for the evolution of only those parts required for the task that they are required. This allows a more flexible design.

In the case of the Khepera robot, when the results obtained in the evolution of the eleven modules in one single stage process are compared with the results obtained by three stages,

we observe that the progressive design of the controllers obtained a slightly better mean fitness value than the mean fitness obtained in the single stage case. Furthermore, the multiple staged approach generated a valid solution 100% of the time, meanwhile the one stage process did 90% of it. Then, progressive design showed to be more stable finding good controllers than the single stage process. The reason is that progressive design does the evolution at small steps in reduced searching spaces, and builds new solutions in new stages starting from an already stable solution provided by the previous stage.

But this fact has a good side and a bad side: the good side is what has been said about building more stable solutions because one stage starts evolving from the last stage stable solution. The bad side of this approach is that only a good enough solution can be provided. Due to the fact that previously evolved modules are freeze from evolving in the new stages, new stages have to carry the solutions found in previous ones. Therefore, it will be very difficult for progressive design to find the best possible controller. Only a good enough controller can be obtained, if a correct evolutionary shape strategy is implemented.

It is not clear whether the progressive design method will be useful in more complex robots with hundreds of modules. Even that progressive design allows the evolution of just a few modules at one stage, in the case of hundreds of modules, the last modules to be evolved will have hundreds of connections to evolve during that stage, what makes the search space large again. It will have to be analyzed in future work if the solution found until that moment will be able to direct the new stage towards a point of the fitness landscape where a solution is near. In both the Khepera and the Aibo experiments, it was observed that the solutions for one stage rapidly evolved from the solutions found in previous stage, manifesting this good landscape starting point effect. This makes us think that the method will be valid for more complex agents, if a progressive enough strategy is performed.

Drawback of the method: it is necessary the use of a simulator, to evolve at least, the first stages until a more or less stable controller is obtained.

6. Conclusion and future work

In this paper we have described the progressive design method for the generation of controllers for complex robots. In the progressive design method, modularity is created at the level of the robot device by creating an independent neural module around each of the sensors and actuators of the robot. This small conceptual modification from functional modularization is the responsible of the reduction of the dimension of the search space and of the bootstrap problem, by allowing a separate evolution of each device (or group of them) by stages.

This special type of staged evolution, evolves the neural controller by stages using evaluation-tasks, which are conditioned to the devices to be evolved. It must be stressed that determining the evaluation-tasks and the set of modules to evolve on each stage is the designer's job, and no general formula is provided. In general, the designer's knowledge of the problem will play a relevant role on it, introducing by hence a bias in the evolutionary process, which we think is unavoidable when working with complex robots. As a drawback, the introduction of knowledge reduces the likelihood of finding an original solution by the evolutionary process may find.

The architecture has been successfully used in several sensory-motor coordinations and compared in performance with other but further experiments show how the architecture would enable its use in more deliberative tasks. In (Télliez & Angulo, 2007), the ability of the

architecture to express its current status is described. The architecture may be used in the future for the complete control of a robot, where the current status is sensed by a superior layer and used to deliberate and modify the robot behavior.

7. References

- Auda, G. and Kamel, M. (1999). Modular neural networks: a survey. *International Journal of Neural Systems*, 9, 2, 129-151
- Bianco, R. and Nolfi, S. (2004). Evolving the neural controller for a robotic arm able to grasp objects on the basis of tactile sensors. *Adaptive Behavior*, 12, 1, 37-45
- J.J. Collins and S.A. Richmond (1994). Hard-wired central pattern generators for quadrupedal locomotion. *Biological Cybernetics*, 71, 375-385
- Davis, I. (1996). A Modular Neural Network Approach to Autonomous Navigation, PhD thesis at the Robotics Institute, Carnegie Mellon University.
- Doncieux, S. and Meyer, J.-A. (2004). Evolution of neurocontrollers for complex systems: alternatives to the incremental approach. *Proceedings of The International Conference on Artificial Intelligence and Applications*.
- Dorigo, M. and Colombetti, M. (2000). *Robot shaping: an experiment in behavior engineering*. The MIT Press
- Elman, J.L. (1991). Incremental learning, or the importance of starting small. *Proceedings of the 13th Annual Conference of the Cognitive Science Society*
- F. Gomez and R. Miikkulainen (1996). Incremental Evolution of Complex General Behavior. Technical report of the University of Texas AI96-248
- Grillner, S. (1985). Neurobiological bases of rhythmic motor acts in vertebrates. *Science*, 228, 143-149
- G. Hornby and J. Pollack (2002). Creating high-level components with a generative representation for body-brain evolution. *Artificial Life*, 8, 223-246
- A.J. Ijspeert (1998). Design of artificial neural oscillatory circuits for the control of lamprey- and salamander-like locomotion using evolutionary algorithms. PhD thesis at the Department of Artificial Intelligence, University of Edinburgh
- Lara, B. and Hülse, M. and Pasemann, F. (2001). Evolving neuro-modules and their interfaces to control autonomous robots. *Proceedings of the 5th World Multi-conference on Systems, Cybernetics and Informatics*
- M.A. Lewis (2002). Gait adaptation in a quadruped robot. *Autonomous robots*, 12, 3 301-312
- Mojon, S. (2004). Using nonlinear oscillators to control the locomotion of a simulated biped robot. Master thesis at École Polytechnique Fédérale de Lausanne
- S. Muthuraman and C. MacLeod and G. Maxwell (2003). The development of modular evolutionary networks for quadrupedal locomotion. *Proceedings of the 7th IASTED International Conference on Artificial Intelligence and Soft Computing*
- Muthuraman, S. (2005). The Evolution of Modular Artificial Neural Networks. PhD thesis at The Robert Gordon University, Aberdeen, Scotland
- Nelson, A. and Grant, E. and Lee, G. (2002). Using genetic algorithms to capture behavioral traits exhibited by knowledge based robot agents. *Proceedings of the ISCA 15th International Conference: Computer Applications in Industry and Engineering*
- S. Nolfi (1997). Using Emergent Modularity to Develop Control Systems for Mobile Robots. *Adaptive Behavior*, 5, 3-4, 343-364

- S. Nolfi and D. Floreano (1998). Coevolving Predator and Prey Robots: Do "Arms Races" Arise in Artificial Evolution?. *Artificial Life*, 4, 4, 311-335
- S. Nolfi and D. Floreano (2000). *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. The MIT Press
- S. Nolfi (2004). *Evolutionary Robotics: Looking Forward*. *Connection Science*, 4, 223-225
- Pfeifer, R. and Scheier, C. (1997). Sensory-motor coordination: the metaphor and beyond. *Robotics and Autonomous Systems*, 20, 157-178
- Pollack, J. B. and Hornby, G. S. and Lipson, H. and Funes, P. (2003). Computer Creativity in the Automatic Design of Robots. *Leonardo*, 36, 2, 115-121
- R. Reeve (1999). Generating walking behaviours in legged robots. PhD thesis of the University of Edinburgh
- R. Reeve and J. Hallam (2005). An analysis of neural models for walking control. *IEEE Transactions in Neural Networks*, 16, 3
- C. W. Seys and R. D. Beer (2004). Evolving walking: the anatomy of an evolutionary search. *Proceedings of the eighth international conference on simulation of adaptive behavior*
- Télliez, R. and Angulo, C. (2007). Acquisition of meaning through distributed robot control. *Proceedings of the ICRA Workshop on Semantic information in robotics*
- Urzelai, J. and Floreano, D. and Dorigo, M. and Colombetti, M. (1998). Incremental Robot Shaping. *Connection Science*, 10, 341-360
- H. Yong and R. Miikkulainen (2001). Cooperative coevolution of multiagent systems. Technical report of the Department of computer sciences, University of Texas AI01-287

Emotional Intervention on Stigmergy Based Foraging Behaviour of Immune Network Driven Mobile Robots

Diana Tsankova
*Technical University – Sofia, Branch Plovdiv
Bulgaria*

1. Introduction

Social insects are simple organisms capable (separately) of very limited activities with a view to intelligent behaviour. Each of them performs a local task unaware both of the behaviour of the others and of the implementation of the global task. However in groups, they possess some degree of intelligence, that allows them to perform extremely complex tasks. These achievements of social insects are due to the phenomenon of stigmergy - a powerful way to coordinate activity over both time and space. The concept of stigmergy has been introduced by the French entomologist Pierre-Paul Grassé in the 1950s during his studies of nest-building behaviour of termites (Grassé, 1959). Stigmergy is derived from the roots "stigma" (goad) and "ergon" (work), thus giving the sense of "incitement to work by the products of work" (Beckers et al., 1994).

Termite nest construction practices are an example of stigmergy. When termites start to build a nest, they impregnate little mud balls with pheromone and place them on the base of a future construction. Termites initially put mud balls in random places. The probability of placing a mud ball in a given location increases with the presence of other mud balls, i.e. with the sensed concentration of pheromone (positive feedback). As construction proceeds, little columns are formed and the pheromone near the bottom evaporates (negative feedback). The pheromone drifting from tops of columns, located near each other, causes the upper parts of the columns to be built with a bias towards the neighboring columns and to join with them into arches (typical building forms).

Corpse-gathering behaviour in ant colonies is another example of a functional and easy coordination through stigmergy. In this case the stigmergic communication is not realized through pheromones but through the corpses themselves. The insects put the corpses of dead nestmates together in a cemetery which is far from the nest. The ants pick ant corpses up, carry them about for a while, and drop them. It seems that ants prefer to pick up corpses from a place with small density of corpses and drop them to a place with higher density. In the beginning there exist a lot of single or small clusters of corpses, but as the time goes on the number of clusters decreases and their size grows up. At the end the process results in the formation of one (or two) large clusters. As it is evident from the two described examples, the ants do not control the overall performance, but rather the environment "puppeteer", the structure that eventually emerges, guides the process.

Stigmergy is an indirect means of communication between multiple agents, involving modifications made to the environment. The agents are programmed so that they obey a simple set of rules and recognize local information to perform a small task. The agent carrying out its task, makes changes in the environment, which stimulates another (or the same) agent to continue working on the task. The environment itself acts as a shared external memory in the context of the system as a whole. The mechanism of stigmergy, combined with environmental physics, provides the basic elements of self-organization. *Self-organization* is a set of dynamical mechanisms whereby structures appear at the global level of a system as a result from interactions among its lower-level components (Bonabeau et al., 1997). However, the relationship between local and global types of behaviour is not easy to understand and small changes at a local level might result in drastic and sometimes unpredictable changes at the global level. Four basic ingredients and three characteristic features (signatures) of self-organization have been identified. The ingredients are: *positive feedback*, *negative feedback*, *amplification of fluctuations* and presence of *multiple interactions*; the signatures are: creation of *spatiotemporal structures* in an initially homogeneous medium, possible attainability of different *stable states*, and existence of parametrically determined *bifurcations* (Bonabeau et al., 1997; Holland & Melhuish, 1999).

Stigmergic concepts have been successfully applied to a variety of engineering fields such as combinatorial optimization (Dorigo et al., 1999; Dorigo et al., 2000), routing in communication networks (Di Caro & Dorigo, 1998), robotics, etc. In robotics, by means of simulated robot teams Deneubourg et al. (1990) have studied the performance of a distributed sorting algorithm (modelling brooding in ant colonies) based on stigmergic principles. Beckers et al. (1994) have extended Deneubourg's work, using physical robots that collect circular pucks into a single cluster, starting from a homogeneous initial environment. The robots have been equipped with two infra-red (IR) sensors, a gripper for pushing objects around, and a switching mechanism, which can sense the local concentration of objects only as below or above a fixed threshold. They have obeyed very simple behavioural rules and have required no capacity for spatial orientation and memory. Holland & Melhuish (1999) have proposed a very similar approach that examines the operation of stigmergy and self-organization in a homogeneous group of physical robots, in the context of the task of clustering and sorting objects (Frisbees) of two different types.

Stigmergy fits excellently into the behaviour-based robot control architecture, which is robust and flexible against the continually changing world. The real-world physics of the environment may be a critical factor for a system level behaviour to emerge. Simulation can provide a picture of possibilities for emergent behaviour. But the use of simulation means that the system is not "grounded" and is unable to exploit the real world physics of the environment. It is for this reason that some authors (Beckers et al., 1994; Holland & Melhuish, 1999) have chosen to implement stigmergic mechanisms directly to behaviour-based robots rather than to undertake any preliminary simulation studies. However, the evolutionary simulation is perhaps the best methodology for the moment for investigating stigmergic phenomena in general, as the real experiments are expensive, time consuming and destructive.

Experiments, similar to those, reported by Beckers et al. (1994), have been repeated in a simulated environment with one robot working alone and two robots working simultaneously in Ref. (Tsankova & Georgieva, 2004). Stigmergy based foraging robots need random movements in order to ensure exploration of all the places of the arena within a

reasonable period of time (Beckers et al., 1994). The problem to solve here is to find a way of speeding up the foraging process, because random movements make the process of formation of the final pile time consuming. Placing simulated detectors for object concentration in order to enhance the perceptive capabilities of the robots is a way of avoiding the loss of time due to wondering in an area without objects, as suggested in the literature (Tsankova et al., 2005; Tsankova et al., 2007). The detectors determine the directions with the maximum and minimum (non-zero) concentrations of pucks (with respect to the robot). The final foraging time has been improved in Ref. (Tsankova et al., 2007) by using two artificial immune networks: one for the navigation control of foraging robots and the other for the object picking up/dropping behaviour. However, the way to be realized the proper detector for object concentration and the accelerating the foraging process - these are still open questions.

For speeding up the foraging process one more time, emotional intervention on the immune navigation control and the object picking up/dropping behaviour is proposed in this research work. It is implemented as a frustration signal coming from an artificial amygdala (a rough metaphor of the natural amygdala, which is situated deep in the brain centre and is responsible for emotions). In a number of studies it has been shown that the psychological factors in general and the emotional factors in particular can be correlated to certain changes in the immunological functions and defense mechanisms (Lazarus & Folkman, 1984; Azar, 2001), i.e. the immune system can be influenced by emotions. This provides a reason for the design of a mixed structure consisting of an innate action selection mechanism, represented by an immune network, and an artificial amygdala as a superstructure over it (Tsankova, 2001; Tsankova, 2007). Another emotional intervention, implemented as an advisor, is applied to the picking up/dropping behaviour mechanism. Depending on the level of frustration the advisor forces the robot, carrying an object, to retain or to drop the object when the robot encounters small or large clusters, respectively. That enhances the positive feedback from the stimulus and speeds up the formation of the final pile.

To illustrate the advantages of the proposed emotional intervention in stigmergy-based foraging behaviour, five control algorithms are simulated in MATLAB environment. They use (respectively): (1) random walks; (2) purposeful movements based on enhanced perception of object concentration; (3) immune network based navigation; (4) emotionally influenced immune network based navigation; and (5) emotional intervention on an immune navigator and on the robot's picking up/dropping behaviour. The comparative analysis of these methods confirms the better performance of the last two of them in the sense of improving the speed of the foraging process.

2. The Task and the Robots

The basic effort in this work is directed toward developing a system of two simulated robots for gathering a scattered set of objects (pucks) into a single cluster (like the corpse-gathering behaviour of ants) and also toward speeding up the foraging process in comparison with the results of similar experiments, reported in the literature. To achieve this task by stigmergy, a simulated robot is designed to move objects that are more likely to be left in locations where other objects have previously been left. The robot is equipped with a simple threshold mechanism - a gripper, able to pick up one puck. An additional detector for puck concentration is used to determine the directions (with respect to the robot) with maximum and minimum (non-zero) concentrations of pucks (Tsankova et al., 2005). This information is

needed to prevent the random walks and to speed up the clustering process. The robots have to pick up pucks from places with small concentration and drop them at places with high concentration of pucks. Five methods of stigmergy based controls are discussed. The first method relies on random walks and codes the stigmergic principles in simple rules with fixed priorities (Beckers et al., 1994; Tsankova & Georgieva, 2004). The other four methods are characterized with enhanced sensing of puck concentration and include (respectively): (1) simple rules with fixed priorities (Tsankova et al., 2005), (2) an immune network for navigation control (Tsankova et al., 2005; Tsankova et al., 2007), (3) emotionally influenced immune network based navigation, and (4) emotional intervention on an immune navigator and on the picking up/dropping behaviour mechanism. The aim is to evaluate the performance of the robots equipped with the above mechanisms and controls in simulations. Before starting each run, 49 pucks are placed in the form of a regular grid in the arena, as shown in Fig.12a. At the beginning of each of the experiments, the robots start from a random initial position and orientation. Every minute of runtime, the robots are stopped, the sizes and positions of clusters of pucks are recorded, and the robots are restarted. The experiment continues until all 49 pucks are in a single cluster. A cluster is defined as a group of pucks separated by no more than one puck diameter (Beckers et al., 1994).

The geometry of robots is shown in Fig.1a, where the radii of the robot and the puck are $R = 0.036 \text{ m}$ and $R_{\text{puck}} = 0.015 \text{ m}$, respectively. Each robot carries a U-shaped gripper with which it can take pucks. The robots are run in a square area $1.5 \text{ m} \times 1.5 \text{ m}$. The robots are equipped with simulated obstacle detectors (five infra-red sensors) and a simulated microswitch, which is activated by the gripper when a puck is picked up. Obstacle detectors are installed in five directions, as shown in Fig.1b. They can detect the existence of obstacles in their directions (sectors S_i , $i = 1, 2, \dots, 5$), and the detecting range of sensors is assumed to be equal to the diameter of the robot. The detectors for puck concentration are located at the same position as the obstacle detectors (Fig.1b). The simulated detector for concentration of pucks can enumerate the pucks (but does not discriminate clusters), which are disposed in the corresponding sector S_i with a range, covering the entire arena. The readings of the detectors for puck concentration are denoted by C_i , $i = 1, 2, \dots, 5$. They are normalized as

$$C_i = N_i^{\text{puck}} / \sum_{j=1}^5 N_j^{\text{puck}}, \quad i = 1, 2, \dots, 5, \quad (1)$$

where N_i^{puck} is the number of pucks, located in the sector S_i .

For the sake of simplicity of simulation the following assumptions in the design of the gripper, the microswitch and the pucks are used (Tsankova & Georgieva, 2004):

- A puck will be scooped only when it fits neatly inside the semicircular part of the gripper.
- If part of a puck is outside of the gripper, the puck will not be scooped, it will not be pushed aside, and the robot will pass across it.
- When the microswitch is activated, the puck may be dropped either on an empty area or on other pucks.
- The pile may grow in height.

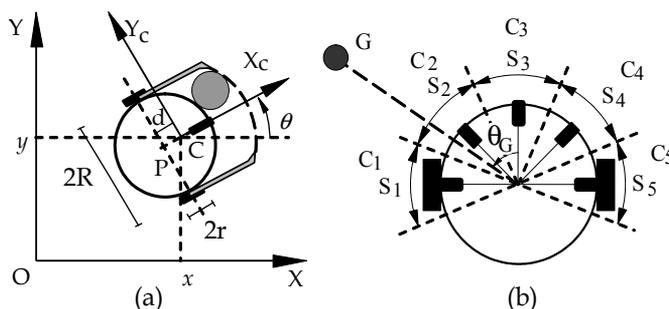


Figure 1. Autonomous mobile robot used in simulations

3. The Algorithms

The five algorithms, mentioned above, are described in more details in this section. The last two of them are the proposed innovations, including an improvement of the robot's navigation and puck picking up/dropping behaviour by introducing an artificial emotional mechanism. The first three algorithms have already been proposed in the literature, and the experiments based on them (implemented in this work) serve as a basis for comparison with the outcomes of the proposed innovations.

I. Stigmergy with random walks

The following rule set is inspired by Ref. (Beckers et al., 1994) and describes the basic behaviours of robots (Tsankova & Georgieva, 2004):

- (1) *If* (there is not a puck in the gripper) & (there is a puck ahead) *then* take one puck in the gripper.
- (2) *If* (there is one puck in the gripper) & (there is a puck ahead) *then* drop a puck, go backward for a while (t_{backward}) and turn at a random angle.
- (3) *If* there are no pucks ahead *then* go forward.
- (4) *If* there is an obstacle (wall or another robot) ahead *then* avoid the obstacle (turn at a random angle and go forward).

Moving in a straight line is the robot's default behaviour, which is executed when no sensor is activated. This behaviour continues until an obstacle is detected or the microswitch is activated (pucks are not detected as obstacles). When the robot detects an obstacle it executes the obstacle avoidance behaviour. On the spot it turns away from the obstacle at a random angle until detectors no longer find out the obstacle, and then goes forward (Beckers et al., 1994). If the robot carries a puck when it encounters the obstacle, the gripper will retain the puck during the turn. The execution of the obstacle avoidance behaviour suppresses the puck dropping one. The threshold of the gripper allows it to take only one puck; more pucks force the microswitch to trigger the puck dropping behaviour. The robot releases the puck from the gripper, goes backwards for a while, and then turns at a random angle, after which returns to its default behaviour and moves forward in a straight line.

II. Stigmergy with enhanced sensing of object concentration

The following set of rules describes the robot's behaviours, when the puck concentration is taken into account (Tsankova et al., 2005):

- (1) *If* (there is not a puck in the gripper) & (there is a puck ahead) *then* take one puck in the gripper.
- (2) *If* (there is one puck in the gripper) & (there is a puck ahead) *then* drop a puck and go backward for a while (t_{backward}).
- (3) *If* (there is not a puck in the gripper) & (there are no pucks ahead) *then* follow the direction, corresponding to the *minimum* (non-zero) reading of the detectors for puck concentration.
- (4) *If* (there is one puck in the gripper) & (there are no pucks ahead) *then* follow the direction, corresponding to the *maximum* reading of the detectors for concentration of pucks.
- (5) *If* there is an obstacle (wall or another robot) ahead *then* avoid the obstacle (turn on the obstacle avoidance behaviour).

When no obstacle detector is activated, the robot executes a goal following behaviour with an artificial goal G (Fig.1b) corresponding to the place with the maximum or minimum concentration of pucks, depending on the presence or absence of a puck in the gripper, respectively. The puck concentration detectors determine the direction of the artificial goal. If all pucks are disposed behind the robot, the low-level control makes the robot turn until a puck concentration detector becomes active. The goal following behaviour continues until an obstacle is detected or the microswitch is activated. The obstacle avoidance and the puck dropping behaviour are the same as the behaviours described in the previous algorithm (Algorithm I).

III. Stigmergy with an immune navigation control

The immune networks for this and for the next control algorithms use enhanced sensing of object concentration. In conformity with the immune navigation control, the set of rules of Algorithm II (from rule (1) to rule (5)) is modified so that *the first two rules remain unchanged*, and the other three are substituted by the following rule (3a) (Tsankova et al., 2005; Tsankova et al., 2007):

- (3a) *If* (there are no pucks ahead) *OR* (there is an obstacle ahead) *then* turn on the collision free goal following behaviour, realized by an *artificial immune network*.

If there is one puck in the gripper, the direction of the goal G is the direction corresponding to the sector with the maximum number of pucks, and if there is no puck in the gripper – the direction with the minimum puck concentration. The immune network implements a collision-free goal following behaviour.

IV. Stigmergy with an emotionally influenced immune navigation control

An artificial emotion mechanism (EM1 in Fig.6) is proposed as a superstructure over the immune network based navigator. It may influence the decision-making mechanism of the immune network, modulating the dynamics of antibody selection that is described in detail in Sections 5 and 6. The control algorithm is the same as Algorithm III, but the rule (3a) is replaced by the following rule (3b):

- (3b) *If* (there are no pucks ahead) *OR* (there is an obstacle ahead) *then* turn on the collision free goal following behaviour, realized by an *emotionally influenced immune network*.

It is expected that the emotional intervention will improve the robot's collision-free goal following behaviour, and therefore it will speed up the foraging process.

V. Stigmergy with two artificial emotion mechanisms

The first of the two artificial emotion mechanisms (EM1) serves for the emotional intervention on the immune navigator as it was described in Algorithm IV. The innovation

here is the second artificial emotion mechanism (EM2) used as an advisor of the puck picking up/dropping mechanism by the regulation output $\gamma_{\text{puck dropping}}^{\text{prohibit}} = 0; 1$ (Fig.7). The following set of rules describes the robot's behaviours, when the two emotion mechanisms are taken into account:

- (1) **If** (there is not a puck in the gripper) & (there is a puck ahead) **then** take one puck in the gripper.
- (2a) **If** (there is one puck in the gripper) & (there is a puck ahead) & ($\gamma_{\text{puck dropping}}^{\text{prohibit}} = 0$) **then** drop a puck and go backward for a while (t_{backward}).
- (2b) **If** (there is one puck in the gripper) & (there is a puck ahead) & ($\gamma_{\text{puck dropping}}^{\text{prohibit}} = 1$) **then** retain the puck and turn on the collision free goal following behaviour, realized by an *emotionally influenced immune network*.
- (3) **If** (there are no pucks ahead) **OR** (there is an obstacle ahead) **then** turn on the collision free goal following behaviour, realized by an *emotionally influenced immune network*.

The emotional advisor of the puck picking up/dropping mechanism in fact influences on the puck dropping behaviour only, as the robot releases the puck under a large frustration level (regulation output of EM2 is $\gamma_{\text{puck dropping}}^{\text{prohibit}} = 0$), and retains the puck when the frustration is small ($\gamma_{\text{puck dropping}}^{\text{prohibit}} = 1$) (Fig.7). The first case corresponds to large puck density, detected by the sensors, and the second - to small density. Due to the dynamics of the amygdala's model (5), the frustration's threshold is different, depending on the direction of robot's movement - towards a larger cluster or in the opposite direction. It is expected that this will enhance the positive feedback from the stimulus (the maximum cluster of objects) and will improve the foraging process in the vicinity of large clusters.

4. Immune Networks

4.1 Biological and artificial immune networks

The human body maintains a large number of immune cells - *lymphocytes*, mainly *T-cells* and *B-cells*. When an *antigen* (a foreign body) invades the human body, only a few of these immune cells can recognize the invader. The *idiotypic network hypothesis*, proposed by Jerne (1974), is based on the concept that lymphocytes are not isolated, but communicate with each other through interaction among antibodies. *B-lymphocytes* have specific chemical structure and produce "Y" shaped *antibodies*. The antibody recognizes an *antigen* like a *key and lock relationship*. The structure of the antigen and the antibody is shown in Fig.2, where the part of the antigen recognized by the antibody is called *epitope*, and the part of the antibody that recognizes the corresponding antigen determinant is called *paratope*. The antigenic characteristic of the antibody is called *idiotope*. Antibodies stimulate and suppress each other by the idiotope-paratope connections and thus form a large-scaled network.

The idiotypic network theory is usually modelled with differential equations simulating the dynamics of lymphocytes. Farmer et al. (1986) have first suggested an abstracted mathematical model of Jerne's *immune network theory*. In robotics Ishiguro et al. (1995b) and Watanabe et al. (1999) have developed a dynamic decentralized behaviour arbitration mechanism based on immune networks. In their approach "intelligence" is expected to emerge from interactions among agents (competence modules) and between a robot and its

environment. A collision-free goal following behaviour has been performed in Ref. (Ishiguro et al., 1995b), and a garbage-collecting problem taking into account self-sufficiency – in Ref. (Watanabe et al., 1999). More detailed surveys of artificial immune systems and their applications can be found in Refs. (Dasgupta & Attouh-Okine, 1997; Garrett, 2005). The description of the dynamics of the antibody selection mechanism and the artificial immune navigator, as they have been presented in Ref. (Tsankova et al., 2007), follows below.

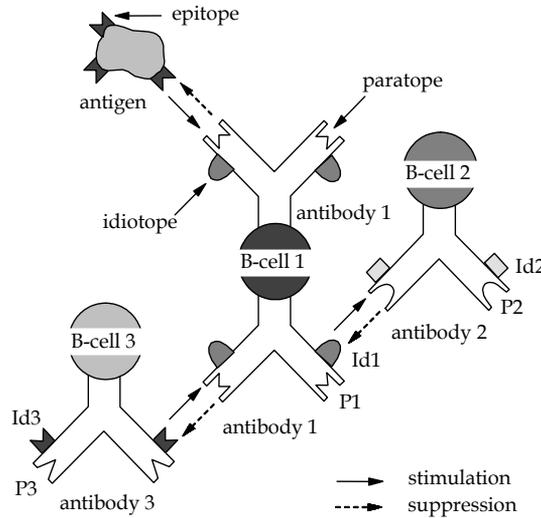


Figure 2. Structure of immune network (Ishiguro et al., 1995b)

4.2 Dynamics of antibody selection mechanism

Consider a goal following and obstacle avoidance navigation task. In such a situation, for example, the distance and direction to the detected obstacle or to the goal work as an antigen, the competence module (simple behaviour/action) can be considered as an antibody, and the interaction between modules is presented as stimulation/suppression between antibodies. The concentration $a_i(t)$ of the i -th antibody is calculated as (Ishiguro et al., 1995a; Ishiguro et al., 1995b):

$$\frac{da_i(t)}{dt} = \left(\frac{1}{N} \sum_{j=1}^N m_{j,i} a_j(t) - \frac{1}{N} \sum_{k=1}^N m_{i,k} a_k(t) + m_i - k_i \right) a_i(t), \quad (2)$$

where N is the number of the antibodies, $m_{j,i}$ and m_i denote affinities between the antibody j and the antibody i , on the one hand, and the antibody i and the detected antigen, respectively. The first and the second terms on the right hand side denote the stimulation and suppression coming from other antibodies, respectively. The third term represents the stimulation coming from the antigen, and the fourth term k_i - the natural death. The affinity coefficients $m_{j,i}$ and m_i are calculated by (Ishiguro et al., 1995b):

$$m_{j,i} = \alpha \sum_k^L I_j(k) \oplus \overline{P_i(k)}, \quad m_i = \beta \sum_k^L E(k) \oplus \overline{P_i(k)}, \quad (3)$$

where α and β are positive constants, \oplus represents the exclusive-or operator, L is the length of the paratope, the idiotope and the epitope, written as binary strings. $I_j(k)$, $P_i(k)$ and $E(k)$ represent the k -th binary value in the idiotope string of the antibody j , the paratope string of the antibody i , and the epitope string, respectively. If the concentration of the antibody exceeds a priori given threshold, the antibody is selected and its corresponding behaviour becomes active towards the world.

4.3 Artificial immune navigator

In this work by "navigator" will be denoted the collision-free goal following behaviour control. The obstacle detectors give binary information 1/0 about the existence or absence of obstacles in their range, respectively. On the basis of the readings of the puck concentration detectors C_i , $i=1,2,\dots,5$ a simulated goal detector can recognize the direction of the goal (maximum/minimum puck heaping) at any position of the obstacle detectors. In the case, in which there is a puck in the gripper, the simulated goal detector responds with 1 to the direction of $C_{\max} = \max(C_i)$ and with 0 to the other four directions. When the robot does not carry a puck, it responds with 1 to the direction of $C_{\min} = \min(C_i)$ and 0 to the rest. Therefore, the robot's simulated detectors discover two types of antigens (obstacle-oriented antigens and goal-oriented ones), and each antigen has a five-bit epitope. The antigens inspire the same two types of antibodies. The antibody's paratope (Fig.3) corresponds to the desirable condition (the precondition, which has to be fulfilled before the activation of the antibody), and its idiotope - to the disallowed antibodies (the antibodies which are impossible or undesirable when the condition of the paratope and its corresponding action are implemented) (Ishiguro et al., 1995b). For mobile robot navigation a simple immune network with 12 a priori prepared antibodies is used (Tsankova & Topalov, 1999; Tsankova et al., 2005) (Fig.4). The first six antibodies are stimulated by obstacle-oriented antigens, and the other six - by goal-oriented ones. Their actions are: move forward (Front), turn right (RS, RM), turn left (LS, LM), move backward (TurnBack). In Fig.4 the goal-oriented paratopes are not presented as binary strings, as they are expressed in calculations, for the sake of the easier explanation of the network. For example, $G \in S_1$ is expressed in calculations by 10000 and denotes that the goal (G) appears in the sector S_1 of the goal detector, and $G \in \text{none}$ - is expressed by 00000, which shows that the goal is not discovered in the five sectors of the goal sensor, i.e. it is behind the robot. The symbol # denotes that the condition can be taken as either 0 or 1, i.e. it can be considered not so important information. Therefore, in (3), when $P_i(k)=\#$ or $I_j(k)=\#$ it determines that $I_j(k) \oplus \overline{P_i(k)} = E(k) \oplus \overline{P_i(k)} = 0.25$. The idiotope includes disallowed antibodies for a situation, in which the paratope condition is fulfilled. For example, the paratope of antibody 9 shows that the goal is discovered in front of the robot in the sector S_3 and the corresponding action is "move forward" (Front). This behaviour will be impossible, if there is an obstacle in front of the robot, i.e. if the obstacle detectors react with the string ##1##, which unites the paratopes of the antibodies 2, 3, 4, 5 and 6, and they are considered to be disallowed. The readings of the puck concentration detectors form the goal-oriented antigens. For example, if the maximum puck heaping has occurred in the sector S_4 , i.e.

$C_{\max} \in S_4$, and the minimum - in S_1 ($C_{\min} \in S_1$), then the epitope string will be 00010 or 10000, corresponding to the availability or absence of a puck in the gripper. In Fig.4 the stimulation connections from the idiotopes to the corresponding paratopes are shown by arrows.

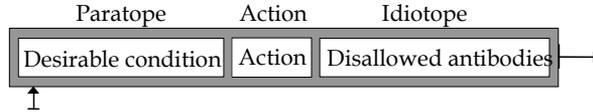


Figure 3. Antibody (Ishiguro et al., 1995b)

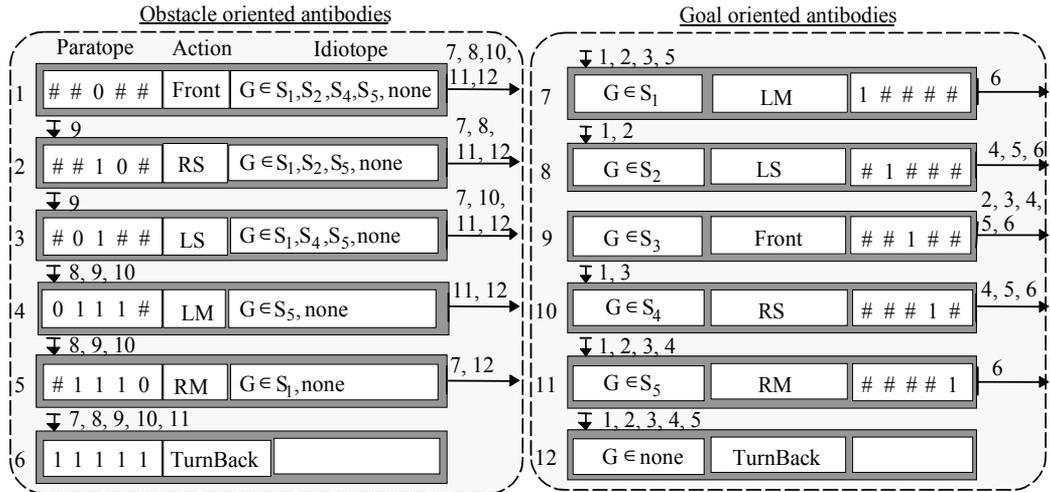


Figure 4. Immune network for collision free goal following behaviour (Tsankova & Topalov, 1999; Tsankova et al., 2005)

For each particular situation detected by sensors, only one of all antibodies wins (in conformity with (2) and (3)) and its action becomes the target behaviour (direction of movement) for the mobile robot. In this work the weight of the two types of behaviour - obstacle avoidance and goal following - is expressed by additional multiplication of the coefficients m_i of the obstacle-oriented antibodies (from 1 to 6) and the goal-oriented antibodies (from 7 to 12) by the weight coefficients k_{goal} and k_{obst} , respectively:

$$m_i = k_{\text{obst}} \beta \sum_{k=1}^L E(k) \oplus \overline{P_i(k)}, \quad i = 1, 2, \dots, 6; \quad m_i = k_{\text{goal}} \beta \sum_{k=1}^L E(k) \oplus \overline{P_i(k)}, \quad i = 7, 8, \dots, 12. \quad (4)$$

5. Emotional Intervention on Immune Network

The emotional intervention on an artificial immune network is inspired by the interactions between immune and emotional systems in living organisms, which have been developed during their struggle to cope with continually changing internal and external environments through hundred millions of years. Today *psychoneuroimmunology* investigates the link between bi-directional communication among the nervous, endocrine, and immune systems and its implications for physical health. In this Section follows: an overview of the

definitions of emotions, models of emotions and their applications, to the purpose of choosing a proper computational model for influence on the artificial immune network designed in the previous Section. At the end, the way of integrating the selected model into the equations of dynamics of antibody selection is described.

5.1 Emotions, models, and applications

Emotion is a key element of the adaptive behaviour, increasing the possibility of survival of living organisms. Science is still looking for a complete definition of emotion. All feelings (states) that affect the survival goal of an agent are called motivational states, such as hunger, thirst, pain, sometimes fear, etc. (Bolles & Fanselow, 1980). Emotions, among other feelings, can change the facial expressions (Descartes, 1989). According to Ekman (1992), there exist six basic emotions: anger, fear, sadness, joy, disgust, and surprise.

One of the most extensively developed low-level neurological models of emotions is that of the amygdala (LeDoux, 1996), especially functioning as a classical fear system of the brain. There exist models that have developed a pure physiological simulation of emotions (emotions described in terms of their physiological reactions) (Picard, 1997), and others that deal with the interactions between emotions (or motivational states), for example, fear and pain (Bolles & Fanselow 1980). The event appraisal models of emotions (Ortony et al. 1988; Rosman et al., 1990) are higher-level psychological models developed to understand the link between events and emotions.

All of the above mentioned and various other computational models of emotions have found application in robotics (Mochida et al., 1995; Breazeal, 2002), affective computing (Picard, 1997), believable ("life-like") agents (Bates, 1992) etc. In robotics Mochida et al. (1995) have proposed a computational model of the amygdala and have incorporated it into an autonomous mobile robot with an innate action selection mechanism based on Braitenberg's architecture No.3c (Braitenberg, 1984). After a brief overview on emotions and their models, the computational version of an amygdala's model seems to be the most convenient for the purposes of the task treated here. A short description of this model follows below.

5.2 Model of the amygdala as an artificial emotion mechanism

The *amygdala* is responsible for the emotions, especially for the most fundamental among them - the fear. It is situated deep in the brain's centre. When the amygdala feels a threat, it mobilizes the resources of the brain and the body to protect the creature from damage. Sensor information obtained by receptors firstly enters the *thalamus*, and then forks into the cerebral *cortex* and the amygdala. Information processing in the cerebral cortex is fine-grained, that is why the signals from the cortex are so slow and refined, and provide detailed information about the stimulus. The signals coming from the thalamus are fast and crude, reaching the amygdala before the signals from the cortex, but providing only general information about the incoming stimulus. The coarse information processing accomplished in the amygdala requires less computing time compared to the one needed by the cortex, since the amygdala just evaluates whether the current situation is pleasant or not. This coarse but fast computation in the emotional system is indispensable for self-preservation of living organisms, which have to overcome the challenges of a continually changing world. The pathways that connect the amygdala with the cortex ("the thinking brain") are not symmetrical - the connections from the cortex to the amygdala are to a large extent weaker

than those from the amygdala to the cortex. The amygdala is in a much better position to influence the cortex. This is one of the reasons for which "the amygdala never forgets (LeDoux, 1996)" and psychotherapy is often such a difficult and prolonged process. Due to the above characteristics, it can be considered that the emotional system regulates activities in the cerebral cortex feed forwardly (Mochida et al., 1995).

In the computational model of the amygdala proposed by Mochida et al. (1995), the emotion of robots is divided into two states: *pleasantness* and *unpleasantness*, represented by a state variable called *frustration*. The neural network representation of this model is shown in Fig.5. Using sensory inputs the level of frustration is formulated as (Mochida et al., 1995):

$$f_{k+1} = \xi_1 \sum_{i=1}^n W_i S_i + (1 + \xi_2 \sum_{i=1}^n W_i S_i - b) f_k, \quad (5)$$

where f_k represents the frustration level of the agent at the moment k , ξ_1 and ξ_2 are coefficients, W_i denotes the weight parameter with respect to the obstacle detector S_i , and b is the threshold, which determines the patience for unpleasantness; n is the number of equipped obstacle detectors; In (5), the first and the second terms on the right hand side denote the frustration levels caused by the direct stimulation of the agent and the recently experienced relationship between the agent and the situation, respectively. The regulation output $\gamma = \gamma(f)$ of the emotional mechanism is determined here as:

$$\gamma = -[1 - e^{-2(f+b_\gamma)}] / [1 + e^{-2(f+b_\gamma)}], \quad (6)$$

where $f \leftarrow f_k$ is taken from (5), b_γ is a bias, and $\gamma \in [-1; 1]$ (Fig.5).

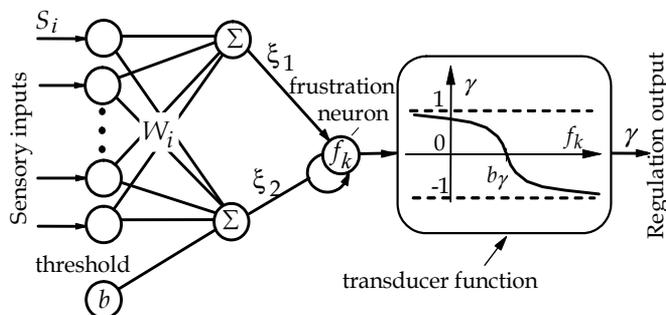


Figure 5. Model of amygdala (Mochida et al., 1995)

5.3 Emotionally influenced dynamics of antibody selection

The emotional intervention on the immune network, whose architecture is shown in Fig.4, can be implemented as a frustration signal coming from the computational model of an amygdala (Fig.5) and influencing the dynamics of the antibody selection mechanism. The regulation output γ of the amygdala is able to modulate different network parameters (affinity coefficients and natural death), or to influence directly the change of concentration of antibodies. Thus it can change the antibody-winner and the final behaviour of the robot.

In the particular navigation problem the emotional mechanism can merely affect the antibodies for goal following behaviour (from 7 to 12) by suppressing the rate of change of the concentrations of those antibodies. This can be obtained by modifying the equations from 7 to 12 of the system of differential equations (2) by multiplying together their right hand side (derivatives of concentrations) and the regulation output γ of the amygdala. Thus, the system of equations (2) is transformed as it follows (Tsankova, 2007):

$$\begin{aligned} \frac{da_i(t)}{dt} &= F_i(a(t), m_i, m_{ij}, k_i), \quad i = 1, 2, \dots, 6, \\ \frac{da_i(t)}{dt} &= \gamma F_i(a(t), m_i, m_{ij}, k_i), \quad i = 7, 8, \dots, 12, \end{aligned} \quad (7)$$

where $F_i(\cdot)$ is the right hand side of (2).

6. Emotional Intervention on Robot's Immune Navigator and on Puck Picking up/Dropping Mechanism

6.1 The navigator

The emotionally influenced immune navigator, proposed in Algorithms IV and V (Section 3), consists of: (1) an immune network (Fig.4) as a basic action selection mechanism; and (2) an artificial emotional mechanism – a model of an amygdala (Fig.5) as a superstructure over the immune network, which modulates the antibody selection. The model of an amygdala (5)-(6) weaves into the differential equations, describing the dynamics of antibody selection (2)-(4) in a way, similar to the one, described in Subsection 5.3. After a number of preliminary experiments with a navigator, which is based on the model (7), that model was modified as follows:

$$\frac{da_i(t)}{dt} = \gamma_{\text{nav}} F_i(a(t), m_i, m_{ij}, k_i), \text{ where } \gamma_{\text{nav}} = \begin{cases} 1, & \text{if } i = 2, 3, \dots, 6, \\ \gamma, & \text{if } i = 1, i = 7, 8, \dots, 12, \end{cases} \quad (8)$$

The modification includes an emotional intervention on both the goal-oriented antibodies from 7 to 12 and the antibody 1, which executes a movement forward when there is not an obstacle in front of the robot. In the absence of obstacles the regulating output of the amygdala has a value of $\gamma = 1$, and thus it does not influence the dynamics of the immune network. However, in the presence of an obstacle, the selection of antibody 1 is manipulated by γ and the probability for this antibody to be selected on a system level (8) decreases. Therefore, in these cases the rotary motion, rather than the rectilinear forward motion is preferred. As a result, more flexible manoeuvring is expected in difficult situations, such as Π -shaped obstacles and narrow passages. A block diagram of the system “emotionally influenced immune navigator – mobile robot” is shown in Fig.6, where α is the “action” part of the antibody winner, and $\mathbf{v} = (v, \omega)$ is the target velocity vector.

The kinematics of a mobile robot with two driving wheels, mounted on the same axis, and a front free wheel is used in simulations (Fig.1a). The motion of the mobile robot is controlled by its linear velocity v and angular velocity ω . The trajectory tracking problem under assumption for “perfect velocity tracking” is posed as in Kanayama et al. (1990) and Fierro

& Lewis (1995). Details of this low-level tracking control are omitted due to the limited space here.

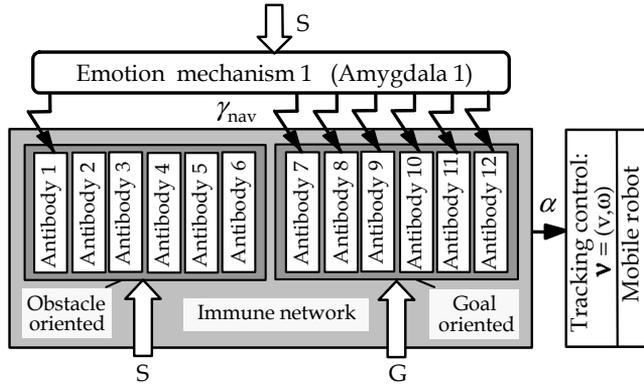


Figure 6. Block diagram of the system 'emotionally influenced immune navigator - mobile robot'

6.2 The puck picking up/dropping mechanism

The idea behind the emotional intervention on the puck picking up/dropping mechanism is to use the frustration threshold of a second amygdala EM2 (Fig.7), whose inputs are the sensor readings of puck concentration, in order to influence the puck dropping behaviour. If a robot with a full gripper collides with a puck ahead (or clusters of pucks), it will drop the puck only if the frustration of the amygdala EM2 exceeds a certain threshold. In the opposite case it will retain the puck and will continue moving in the same direction. Since the robot does not perceive the pucks as an obstacle, it does not go round them, but passes across them. It is assumed, that the matter in the robot's hand is a single puck or a very small cluster of pucks, since the frustration is below the threshold. So, the stigmergic process will rather benefit than be harmed by the destruction of the small cluster (if this occurs). Due to the dynamics of the amygdala's model, the frustration threshold is different, depending on the direction of the robot's movement - towards a larger cluster or in the opposite direction. The regulation output of EM2 generates the following signal:

$$\gamma_{\text{puck dropping}}^{\text{prohibit}} = \begin{cases} 0, & \text{if } \gamma \leq 0, \\ 1, & \text{if } \gamma > 0. \end{cases} \tag{9}$$

where the values '0' and '1' mean 'permission' and 'prohibition' of the puck dropping behaviour, respectively. A block diagram illustrating EM2 is shown in Fig.7.

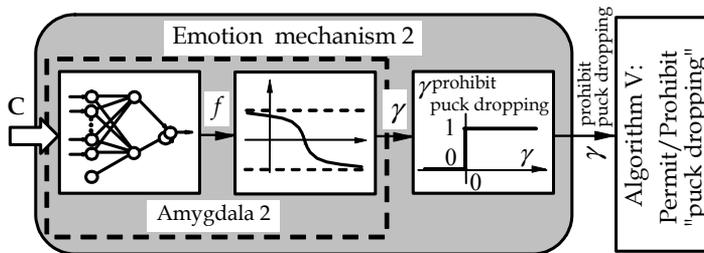


Figure 7. Block diagram of the emotional intervention on puck dropping behaviour

7. Simulation Results and Discussions

7.1 Simulation experiments with emotionally influenced navigator

The improvement of the navigation of a single robot should be aimed at increasing the reliability and the efficiency of work of the overall robot group in the puck foraging task. The simulation experiments in this Subsection emphasize the emotionally affected immune navigator; they make clear both the mechanism of action of amygdala's model and the way by which it influences the immune navigator; show the advantages of the mixed structure by a comparative analysis carried out between the proposed emotionally influenced immune navigator and the following control structures: (1) Braitenberg's vehicle No.3c with an emotional mechanism (Mochida et al., 1995), and (2) the immune network based navigator.

The simulation experiments were implemented in MATLAB environment with the sampling time set to $T_o = 0.01$ s. The parameters of the trajectory tracking controller were chosen to correspond to a critical dumping case (Kanayama et al., 1990), and the reference linear and angular velocities were $v_{ref} = 0.3$ m/s and $\omega_{ref} = 0$ rad/s, respectively. The duration of the movement backward was $t_{backward} = 0.75$ s. The parameters needed for modelling the dynamics of the antibody selection were heuristically determined as: $\alpha = \beta = 1$; $k_{obst} = 1$ and $k_{goal} = 0.4$; the death rate $k_i = 0.1$, $i = 1, 2, \dots, 12$; and the threshold for the antibody selection mechanism $a_{thresh} = 0.33$. The following values for the angles of the immune network's "action" part were experimentally chosen: $Front = 0$ rad, $LS = -RS = 0.65$ rad, $LM = -RM = 1.4$ rad, and $Back = 1.4$ rad. The parameters of the amygdala's model have been investigated theoretically and experimentally in (Tsankova, 2001) and the values, determined as proper there, were used in this study, i.e.: $\xi_1 = 0.003$, $\xi_2 = 0.3$, $b = 0.12$, $b_\gamma = 5$ and $\mathbf{W} = (0.25 \ 0.75 \ 2.5 \ 0.75 \ 0.25)^T$. The frustration signal was limited in accordance with its definition domain $f \in [f_{min}; f_{max}]$, where $f_{min} = 0$ and $f_{max} = 18$ (Tsankova, 2001).

The robot, controlled by the immune navigator without emotional intervention, was simulated, and the results from the simulation are shown in Fig.8. The symbols $*$, \times and \circ denote the goal position, as well as the initial and final position of the robot, respectively. The robot orientation with respect to the inertial basis is denoted by θ . The measuring units on the two axes of the robot's rectangular work area are metres. Different behaviours were obtained by suppressing some antibodies of the immune network shown in Fig.4. For example, obstacle avoidance behaviour (Fig.8a) was derived by suppressing the goal-oriented antibodies from 7 to 12, and for achieving wall following behavior (Fig.8b) the antibody 1 was additionally suppressed. As a result of suppressing the obstacle-oriented antibodies from 1 to 6, goal following behaviour was evoked (Fig.8c). When all 12 antibodies interacted (without any external inhibition) the resultant behaviour was a collision-free goal following behaviour (Fig.8d).

Fig.9a shows another simulation result of a robot with an immune navigator without emotional intervention. Although all antibodies interact without any external inhibition, the robot sometimes gets stuck in small corners, dead ends, and narrow passages. Fig.9b illustrates a better performance of the robot in the same environment, when it is equipped with the proposed emotionally affected immune navigator. On the basis of amygdala's

frustration level f the regulation output γ_{nav} influences the immune network by suppressing, stopping or reversing the goal following behaviour, and thus focuses the attention on the avoidance of obstacles in critical situations. The amygdala (in high frustration) stimulates wall following behaviour rather than obstacle avoidance (the antibody 1 is suppressed simultaneously with the goal following antibodies from 7 to 12) (Fig.6). This behaviour proves to be more successful than the obstacle avoidance in respect to increasing the probability for overcoming some closed-loop situations. For easier understanding of the effect of emotional intervention on the immune navigator, the transitions of the activity of frustration and the regulation output of the amygdala in the case of Fig.9b are calculated. The results are shown in Fig.10, where the letters from "A" to "J" correspond to those in Fig.9b. The frustration level increases when the robot reaches an impasse, and the goal following activity decreases. The robot could resign the goal pursuing. The maximum possible value of frustration $f_{max} = 18$ (or near it) in the intervals "AB", "CD", "EF", "GH", and "IJ" corresponds to a "near-dead-end" situation (the obstacle is exactly between the robot and the goal, in front of the robot, in the role of a bracket) or a "narrow passage" situation.

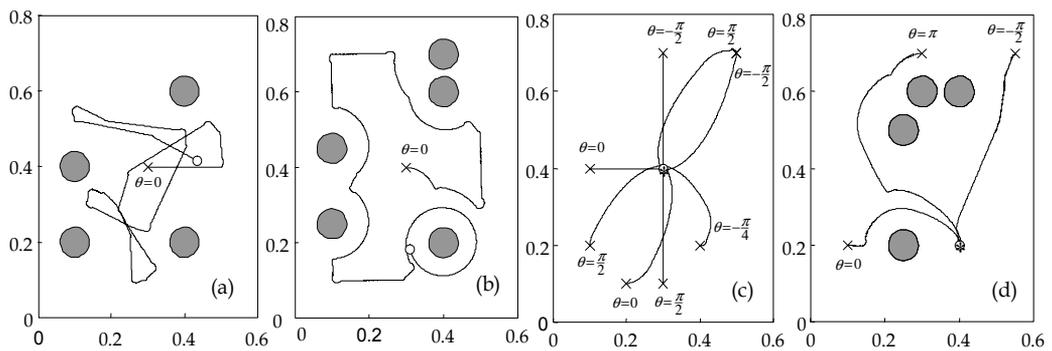


Figure 8. Obstacle avoidance (a), wall following (b), goal following (c), and collision-free goal following (d) behaviours

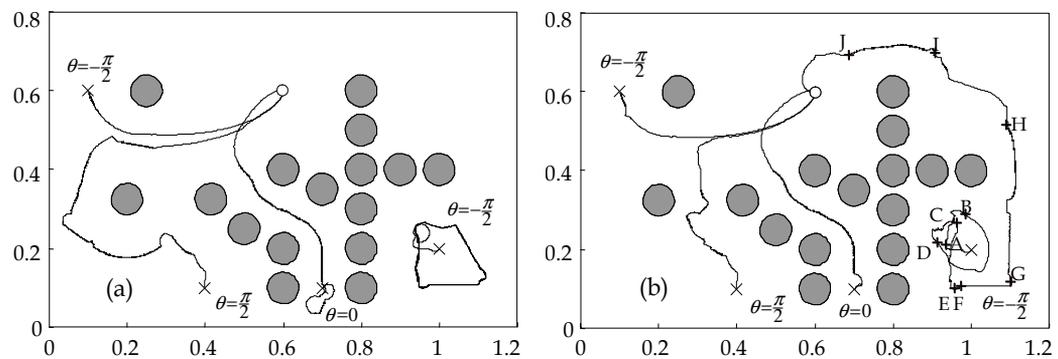


Figure 9. Immune navigator: (a) - independently acting, and (b) - emotionally influenced

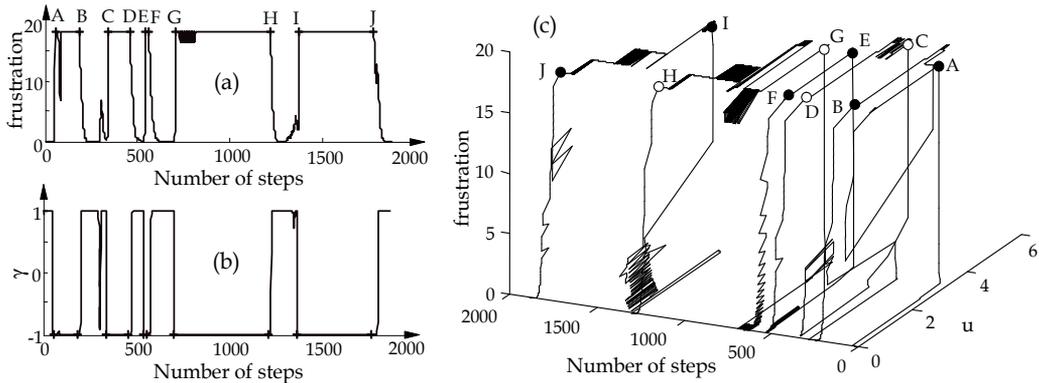


Figure 10. Transition of frustration/regulation output of amygdala used in simulations in Fig.9b

Consider the weighted sum of sensor inputs of the amygdala's model (5) $u(t) = \sum_{i=1}^n W_i S_i(t)$

(or $u(t) = \sum_{i=1}^n W_i C_i(t)$ for the case of EM2). Since the sensor signals are normalized between 0

and 1 inclusive ($S_i = 0; 1$ ($C_i \in [0; 1]$), $i = 1, \dots, 5$), then $u \in [0; U_{\max}]$, where $U_{\max} = \sum_{i=1}^n W_i$.

The change of the frustration signal of the amygdala (5) versus the simultaneous time (number of steps) and the weighted sum of sensor readings u is shown in Fig.10c. In this figure the intervals marked with the same colour "o" sign correspond to the high frustration level of the amygdala. When u increases, the value of u in which f changes drastically (the threshold) is higher than it is when u decreases. This hysteresis (Fig.10c) is due to the dynamics of the amygdala (5). Therefore, the amygdala remembers for a certain period of time "the fear of encountered obstacles" (or the enhanced puck concentration - for EM2 in Algorithm V). The immune navigator can benefit from the short-term memory of the amygdala in a mixed structure composed of the immune network and the emotional mechanism.

In case of absence of obstacles the artificial amygdala does not influence the action selection mechanism, because $\gamma = 1$ and equation (7) is the same as (2). When obstacles are present the goal following behaviour is switched off ($\gamma = 0$) and the robot focuses attention on the avoidance of obstacles (wall following behaviour or obstacle avoidance). The value $\gamma = -1$ usually corresponds to hard situations when a large obstacle is situated between the robot and the goal. In this case the antibody (from the goal-oriented antibodies), corresponding to the goal direction, has the least probability to win. However, since the detected obstacle is possibly situated in this direction too, the avoidance of the obstacle will be facilitated. Thus in a critical situation the robot forgets about the goal that could cause it to get stuck, and focuses attention on the avoidance of obstacles in order to get out of the impasses. Besides, the emotion mechanism EM1 provides the immune navigator with an additional (small amount of) memory about the obstacles recently met. The navigation becomes more careful, which helps the robot to avoid getting stuck.

Additionally, thirty experiments in three different environments (as shown in Fig.11), were carried out with the mobile robot being equipped with the following navigators: (1) emotionally influenced Braitenberg's architecture No.3c, (2) artificial immune network, and (3) emotionally affected immune network. The structures and parameters of the first navigator were the same as in Refs. (Tsankova, 1999; Tsankova, 2001). In each sample, the robot started from a random initial position and orientation, and had to reach the goal, which was also at a random position. The success ratio of the emotionally affected immune navigator was higher than that of the others (87% vs. 60% (emotionally affected Braitenberg's vehicle No.3c) and 77% (immune navigator without emotional intervention)). At the same time the emotionally affected immune navigator was faster (the average time in steps) about 1.16 and 1.38 times in comparison to the others, respectively (Table 1). The emotionally affected immune navigator had better manoeuvring in narrow passages and Π -shaped obstacles (a very difficult test for the agents with local vision and reactive behaviour, because it often causes an impasse) than the others. Perhaps this is due to the artificial amygdala's property to function as a short-term memory for the fear of encountered obstacles. Besides, the way in which the emotion mechanism EM1 is connected to antibodies reinforces the emerging of wall following behaviour (very useful for overcoming impasses).

The very good performance of the emotionally influenced immune navigator compared with the other two intelligent navigators considered above confirmed the reason to use the emotional intervention for navigation purposes and to transfer it to puck picking up/dropping behaviour control in a stigmergy based foraging task.

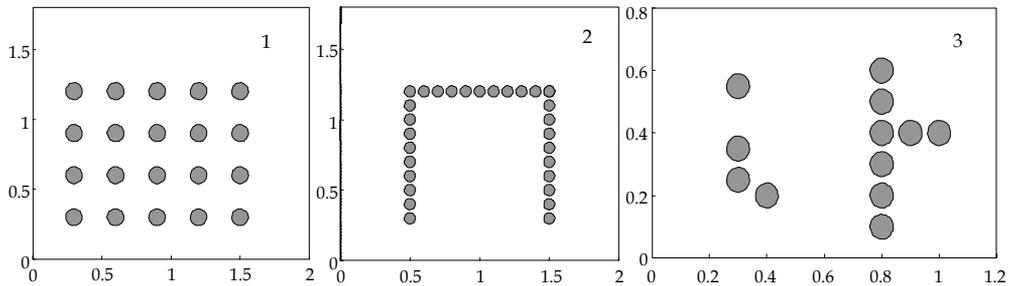


Figure 11. Experimental environments

Emotionally affected Braitenberg No. 3c		Immune network navigator		Emotionally affected immune navigator	
Success [%]	Time [steps]	Success [%]	Time [steps]	Success [%]	Time [steps]
60	2063	77	2460	87	1784

Table 1. Simulation results. Each navigator is presented by the average values from thirty experiments, distributed in the environments shown in Fig.11

7.2 Experiments with the five control algorithms

The five control algorithms described above in Section 3 were simulated in MATLAB environment. The use of the first three of them gives a basis for comparison with the last two algorithms, whose purpose is to improve the speed of the puck foraging process. The expectations are that speeding up is to be achieved by: (1) improvement of the collision-free goal following behaviour by EM1, and (2) reinforcement of the positive feedback from the

stimuli (minimum/maximum puck concentration) by both EM2 and EM1. The foraging task included gathering pucks in a pile by one robot, working alone, and by two robots working simultaneously.

Simulations of puck gathering behaviour by two robots with the first control algorithm (with random walks) are shown in Fig.12. As Beckers et al. (1994) have specified, the experiments have three more or less distinct phases, regardless of the number of robots. In the beginning there are only single pucks on the arena (Fig.12a). In the first phase, a robot moves forwards scooping one puck into the gripper. When two pucks have been gathered, the robot drops them, leaving them as a cluster of two, and moves off in another direction. Shortly after that most pucks are in small clusters with less probability to be destroyed (Fig.12b). In the second phase, the robot removes one puck from clusters by striking the clusters at a certain angle with the gripper. The puck removed in this way is added to other clusters when the robot collides with them. Some clusters increase rapidly at this phase and after some time there is a small number of relatively large clusters (Fig.12c). The third and most prolonged phase consists of the occasional removal of a puck from one of the large clusters, and the addition of this puck to one of the clusters, often to the one it had been taken from (Fig.12d,e). As a result, a single cluster is formed (Fig.12f). Spatiotemporal structures appear in an initially homogeneous environment, and one of the signs of self-organization is present.

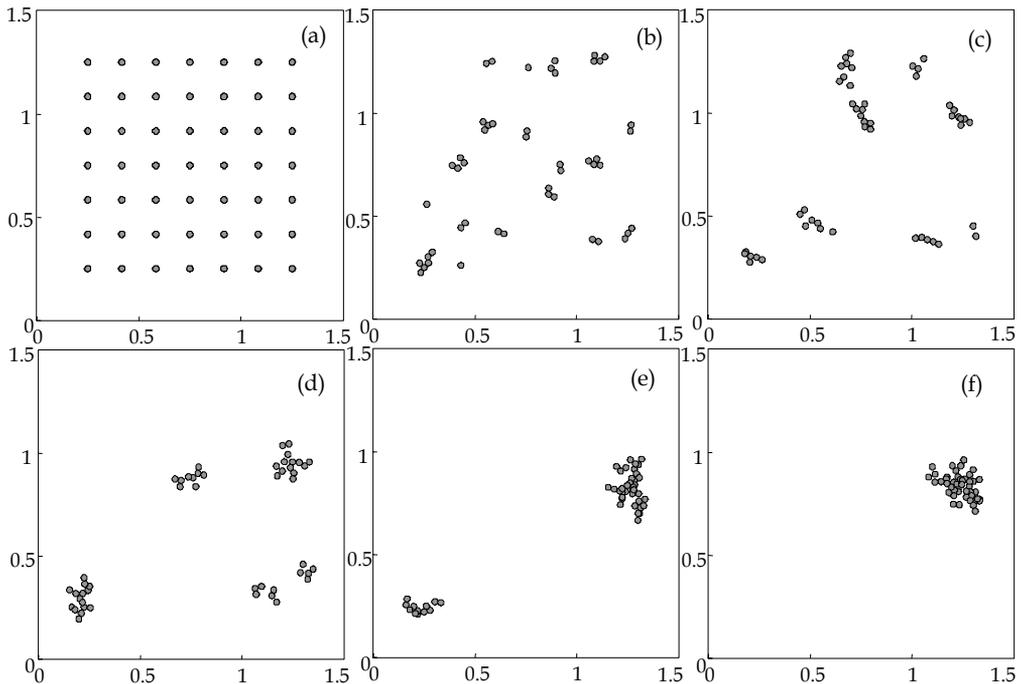


Figure 12. The initial setup (a) and time evolution of a foraging experiment involving two robots. Phase I (b) is characterized by a large number of small clusters consisting of 1 to 6 pucks. In phase II (c) some clusters grow rapidly. Phase III (d, e) includes competition between a small number of large clusters and leads to gathering of all pucks in one pile (f)

The puck dropping mechanism recognizes only a predetermined threshold of puck density - two pucks. It cannot differentiate between the local concentration of two pucks and more than two pucks. Stigmergic mechanism organizes the transfer of pucks from smaller to larger clusters, although the robots under the first control algorithm are unable to discriminate between them with their sensors. The cluster gains pucks from more "frontal" collisions of the robots with it, and loses pucks from almost tangential collisions. The probability for a frontal or tangential collision with a random cluster to be produced depends on the size, shape and position of the cluster. Larger clusters are more likely to gain pucks and less likely to lose pucks than smaller clusters. Due to the constant number of pucks in the environment, in the end all the pucks will be gathered in a single cluster. If the experiment goes on, a puck will occasionally be removed from this single cluster, but it will be returned to it as there is no other pucks in the environment which can trigger the puck dropping behaviour.

In the second control algorithm the random walks are replaced by purposeful moves, taking into account the perceived maximum/minimum concentration of pucks. Under the direction of the place with maximum puck density the robot assumes the direction, corresponding to the sector S_i , whose reading C_i has counted the most of pucks (it makes no difference whether they are situated in one cluster or not). The direction of the place with minimum (non-zero) puck density is determined in a similar way. The robot with an empty gripper goes to the place with minimum (non-zero) concentration of pucks, scooping one puck into the gripper. If there is a puck in the gripper, the robot turns to the place with the maximum pucks and goes forward. The substitution of the random walks with purposeful moves does not violate the stigmergic principles, and only suppresses time-consuming wandering around in an area without pucks. In the third control algorithm the immune navigator implements a collision-free goal following behaviour. The goal directions are the directions with maximum and minimum puck concentrations, depending on the availability or the absence of a puck in the gripper, respectively. However, since the immune network makes decision at the system level, it can assign to the robot a target direction, different from the minimum and maximum puck heaping directions. The innovation in the fourth control algorithm - the emotional intervention (by EM1) on the immune navigator, leads to some improvement of the collision-free goal following behaviour, as it was discussed in the previous Subsection 7.1. That includes more successful avoidance of obstacles (the boundary of working area and the other robot) and also tracking the directions of the maximum/minimum puck concentration.

The advisor (EM2) for the puck dropping behaviour is the basic innovation in the fifth control algorithm. In fact advices are followed unconditionally as a permission or prohibition of this behaviour. If the conditions for puck dropping are available, then, in comparison with the previous algorithms (from I to IV), here EM2 prohibits dropping the puck if its regulation output $\gamma_{\text{puck dropping}}^{\text{prohibit}} = 1$, which can occur when the robot moves from a place with a lower puck concentration to a place with a higher one and the frustration is low (u is under a certain threshold). This prevents the robot from dropping a puck when it encounters a single puck or a small cluster, during motion towards a larger cluster. The robot will drop the retained puck most probably into that large cluster (where $\gamma_{\text{puck dropping}}^{\text{prohibit}} = 0$). However, in the opposite direction, the threshold is lower (because of the hysteresis, which was shown in the example, illustrated in Fig.10c). Therefore, if there is a

puck taken from the large cluster, the robot will drop it in the immediate vicinity of the cluster (in the frame of the frustration's hysteretic zone). That may decrease the probability of destroying the large cluster. The two actions speed up the formation of large clusters. Therefore, EM2 reinforces once again the positive feedback from stimuli, giving the global perception of puck concentration certain superiority to the local one. And the result is present - speeding up the puck foraging process, which is evident from Fig.13, illustrating the time evolution of a foraging experiment involving two robots with the same control algorithm. The parameters of the Amygdala 2 in EM2 were the same as those of the Amygdala 1 (EM1), except for the upper limiting value of frustration, which here was set to be $f_{\max} = 9$.

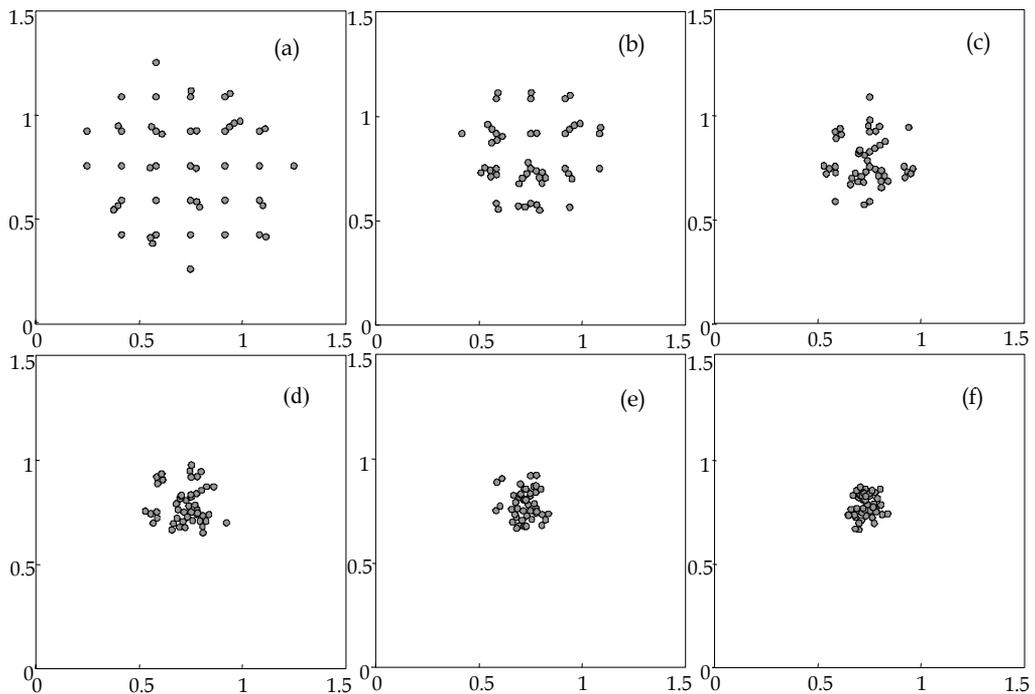


Figure 13. The time evolution of a foraging experiment involving two robots with the two emotional mechanisms EM1 and EM2 (Algorithm V): (a) 1 min, (b) 2 min, (c) 3 min, (d) 4 min, (e) 5 min, and (f) 7 min

Fig.14 illustrates the behaviour of one robot working alone for 1.5 min duration, starting from the initial setup of pucks under a navigation algorithm based on: (a) random walks, and (b) emotionally influenced immune network. The robot with the emotionally influenced immune navigator carried out a specific circular movement, foraging pucks from the periphery to the centre (Fig.14b) and thus forming a central pile (Fig.13f). This effect has been observed in simulations involving a robot equipped only with an immune navigator (Tsankova et al., 2007). Maybe this is due to the *wall following emergent behaviour*, which appears under immune network navigation control and which is reinforced by the EM1.

The simulation experiments with the five control algorithms, involving either one robot working alone, or two robots working simultaneously, led to gathering all pucks in a single pile, located in a random place. The two robots working simultaneously finished the task faster than the solitary one. The average values of (1) the number of clusters and (2) the maximum cluster size from the four experiments with the five control algorithms are shown in Fig.15. The results from these experiments are given in Table 2. The performance of the algorithms is assessed on the basis of the foraging time. The assessment values are the average values of the final foraging time from the four experiments with one solitary working robot and two robots working simultaneously. It is evident that the foraging time decreases drastically from the first to the fifth control algorithm. The achievement of the two robots equipped with the two emotional mechanisms EM1 and EM2 (Algorithm V) and working simultaneously is the best. Unfortunately, the proposed control method, represented by Algorithm V, as the other ones (except for the method, using random walks), relies on a simulated detector for puck concentration, whose physical realization is still an open issue.

Algorithms	Time [min]	
	1 Robot	2 Robots
I. Stigmergy with random walks	1174	705
II. Stigmergy with enhanced sensing of puck concentration	189	109
III. Stigmergy with an immune navigation control	18	13
IV. Stigmergy with an emotionally influenced immune navigation	15	9
V. Stigmergy with two artificial emotion mechanisms	10	7

Table 2. Performance of the five control algorithms, described in Section 3. The foraging time is presented by the average value from four experiments

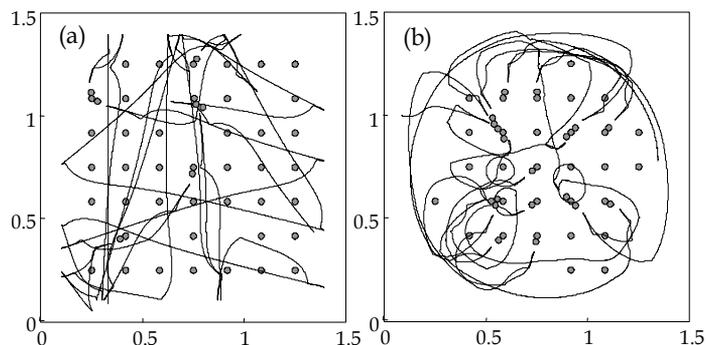


Figure 14. Simulation of a foraging behaviour of one robot working alone for 1.5 min duration, starting from the initial setup of pucks under navigation algorithm using: (a) random walks, and (b) emotionally influenced immune navigator

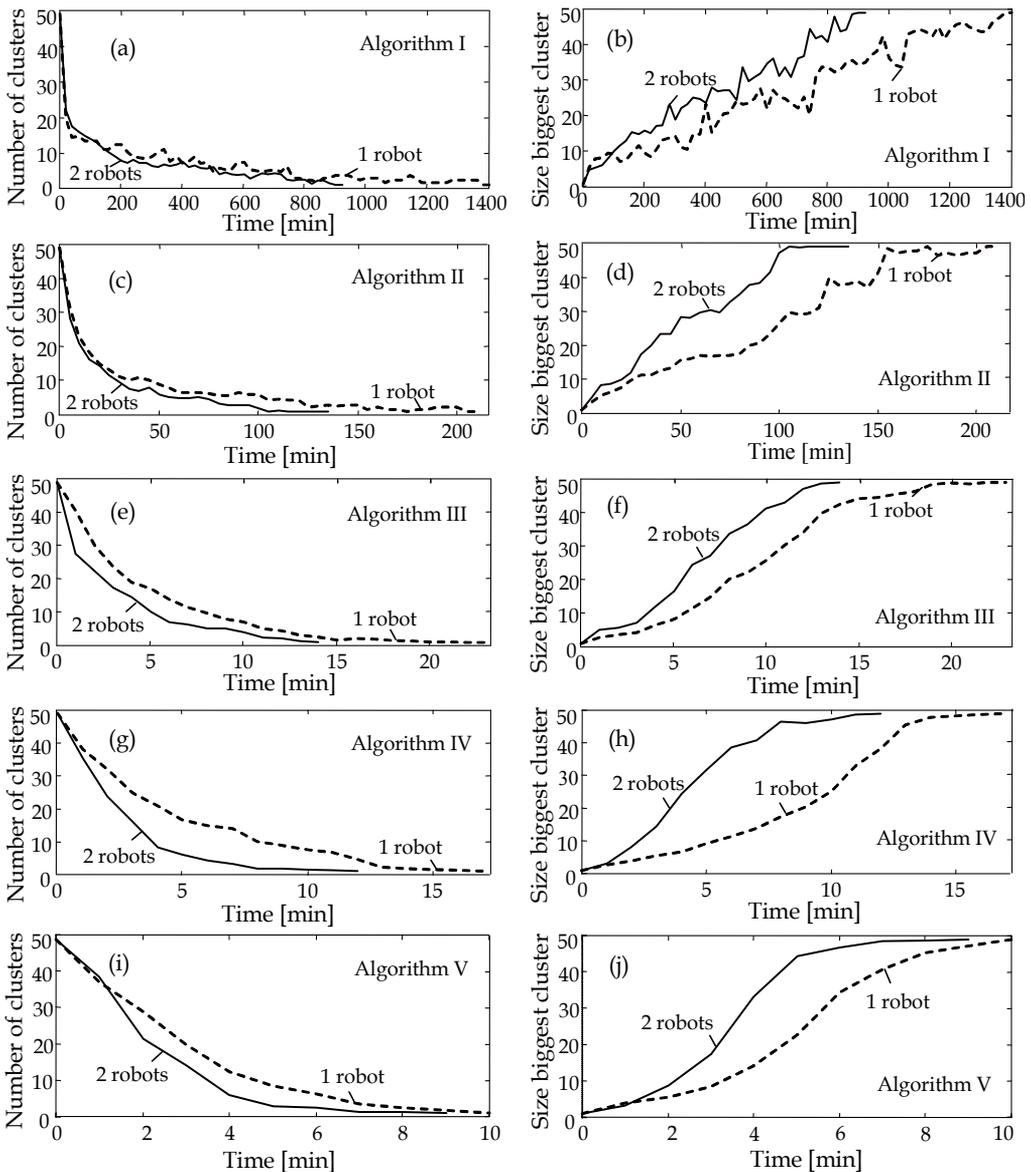


Figure 15. Time evolution of the number of clusters and the size of the biggest cluster for experiments with the five algorithms

8. Conclusion

The proposed stigmergy based foraging behaviour control using two artificial emotion mechanisms – one as a superstructure over the immune navigator, and another as an advisor of puck picking up/dropping behaviour, improves the speed of the clustering process. The intervention of the EM1 on the immune navigator improves the collision-free goal following behaviour of each robot, which affects the final implementation of the

foraging task. Besides, it enhances the positive feedback from the stimuli, since it improves the tracking of the direction of sensor reading for the minimum/maximum puck concentration. The intervention of the EM2 on the puck dropping behaviour reinforces once more the positive feedback from the places with maximum puck concentration. Future work will include the use of optimization techniques for parameter tuning of the amygdala model of the emotional mechanisms EM1 and EM2, as well as of the immune network. It is also necessary to experiment with the proposed emotionally influenced control on real immune network driven robots implementing a puck foraging task. This brings up the question of the physical realization of a detector, resembling the simulated sensor for puck concentration.

9. References

- Azar, B. (2001). A new take on psychoneuroimmunology. *Monitor on Psychology*, Vol.32, No.11, December 2001
- Bates, J. (1994). The role of emotion in believable agents. *Communications of the ACM*, Vol.37, No.7, July 1994, pp.122-125
- Beckers, R.; Holland O.E. & Deneubourg, J.L. (1994). From local actions to global tasks: Stigmergy and collective robotics. *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems Artificial Live IV*, Brooks, R. & Maes, P., (Eds.), pp. 181-189, MIT Press, Cambridge, MA
- Bolles, R.C. & Fanselow, M.S. (1980). A perceptual defensive recuperative model of fear and pain. *Behavioral and Brain Sciences*, Vol. 3, pp. 291-301
- Bonabeau, E.; Theraulaz, G.; Deneubourg, J.L., Aron, S. & Camazine, S. (1997). Self-organization in social insects. *Trends in Ecology and Evolution*, Vol.12, No.5, May 1997, pp.188-193
- Braitenberg, V. (1984). *Vehicles: Experiments in Synthetic Psychology*, MIT Press, Cambridge, MA
- Breazeal, C.L. (2002). *Designing sociable robots*, MIT Press, Cambridge, MA
- Dasgupta, D. & Attoh-Okine N. (1997). Immunity-based systems: A survey. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pp.363-374, Orlando, Florida, 12-15 October, 1997
- Deneubourg, J.L.; Goss, S.; Franks, N.R.; Sendova-Franks, A.; Detrain, C. & Chretien, L. (1990). The dynamics of collective sorting: robot-like ants and ant-like robots. *Simulation of Adaptive Behaviour: from animals to animats*, Meyer, J-A. & S. Wilson, S., (Eds.), pp. 356-365, MIT Press, Cambridge, MA
- Descartes, R. (1989). *The Passions of the Soul*, Trans. Stephen Voss, Hackett Publishing Company, Cambridge
- Di Caro, G. & Dorigo, M. (1998). AntNet: Distributed stigmergic control for communication networks. *Journal of Artificial Intelligence Research*, Vol.9, pp.317-365
- Dorigo, M.; Di Caro, G. & Gambardella, L.M. (1999). Ant algorithms for discrete optimization. *Artificial Life*, Vol.5, No.2, April 1999, pp.137-172
- Dorigo, M., Bonabeau, E. & Theraulaz, G. (2000). Ant algorithms and stigmergy. *Future Generation Computer Systems*, Vol.16, No.8, pp.851-956
- Ekman, P. (1992). Facial expressions of emotion: New findings, new questions. *Psychological Science*, Vol.3, No.1, pp.34-38

- Farmer, J.; Packard, N. & Perelson, A. (1986). The immune system, adaptation and machine learning, *Physica D*, Vol. 22, pp.187-204
- Fierro, R. & Lewis, F.L. (1995). Control of a nonholonomic mobile robot: backstepping kinematics into dynamics. *Proceedings of the 34th Conference on Decision&Control*, pp.3805-3810
- Garrett, S.M. (2005). How do we evaluate artificial immune systems? *Evolutionary Computation*, Vol.13, No.2, pp.145-178
- Grassé, P.P. (1959). La reconstruction du nid et les coordinations inter-individuelles chez *Bellicositermes natalensis* et *Cubitermes* sp. La theorie de la stigmergie: Essai d'interpretation des termites constructeurs. *Insectes Sociaux*, Vol. 6, pp. 41-83
- Holland, O. & Melhuish, Ch. (1999). Stigmergy, self-organization, and sorting in collective robotics. *Artificial Life*, Vol. 5, No. 2, pp. 173-202
- Ishiguro, A.; Kondo, T.; Watanabe, Y. & Uchikawa, Y. (1995a). Dynamic behavior arbitration of autonomous mobile robots using immune networks. *Proceedings of ICEC'95*, pp.722-727
- Ishiguro, A.; Watanabe, Y. & Uchikawa, Y. (1995b). An immunological approach to dynamic behavior control for autonomous mobile robots. *Proceedings of IROS'95*; Vol.1, pp.95-500
- Jerne, N.K. (1974). Towards a network theory of the immune system. *Annals of Immunology*, Vol.125C, pp.373-389
- Kanayama, Y.; Kimura, Y.; Miyazaki, F. & Noguchi, T. (1990). A stable tracking control method for an autonomous mobile robot. *Proceedings of IEEE International Conference on Robotics and Automation*, Vol.1, pp.384-389
- Lazarus, R.S. & Folkman, S. (1984). *Stress, Appraisal, and Coping*, Springer, New York
- LeDoux, J. (1996). *The Emotional Brain*, Simon & Schuster, New York
- Mochida, T.; Ishiguro, A.; Aoki, T. & Uchikawa, Y. (1995). Behaviour arbitration for autonomous mobile robots using emotion mechanisms. *Proceedings of IROS'95*, Vol.1, pp.516-521
- Ortony, A.; Clore, G. & Collins, A. (1988). *The Cognitive Structure of Emotions*, Cambridge University Press, Cambridge
- Picard, R. W. (1997). *Affective Computing*, MIT press, Cambridge, MA
- Rosman, I.J.; Jose, P.E. & Spindel, M.S. (1990). Appraisals of emotion-eliciting events: testing a theory of discrete emotions, *Journal of Personality and Social Psychology*, Vol.59, No.5, pp. 899-915
- Tsankova, D.D. & Topalov, A.V. (1999). Behaviour arbitration for autonomous mobile robots using immune networks. *Proceedings of the 1st IFAC Workshop on Multi-Agent Systems in Production (MAS'99)*, pp. 25-30, Wien, Austria, 2-4 December, 1999
- Tsankova, D. (2001). *Artificial Immune Networks and Emotion Mechanisms for Control of a Class of Autonomous Agents*, Ph.D. Thesis, TU-Sofia, Branch Plovdiv, Bulgaria
- Tsankova, D.D. & Georgieva, V.S. (2004). From local actions to global tasks: Simulation of stigmergy based foraging behavior. *Proceedings of the 2nd International IEEE Conference "Intelligent Systems"*, Vol.1, pp.353-358, Varna, Bulgaria, 22-24 June, 2004
- Tsankova, D.; Georgieva, V.; Zezulka, F. & Bradac, Z. (2005). Immune navigation control for stigmergy based foraging behaviour of autonomous mobile robots. *Proceedings of the 16th IFAC World Congress*, Prague, Czech Republic, 4-8 July, 2005

- Tsankova, D.; Georgieva, V.; Zezulka, F. & Bradac, Z. (2007). Immune network control for stigmergy based foraging behaviour of Autonomous Mobile Robots, *International Journal of Adaptive Control and Signal Processing*. Special Issue on "Autonomous and adaptive control of vehicles in formation", Veres, S.M.. (Ed.), Vol. 21, No. 2-3, pp. 265-286, John Wiley & Sons
- Tsankova, D. (2007). Emotional intervention on an action selection mechanism based on artificial immune networks for mobile robot navigation, *Adaptive Behavior*, Submitted.
- Watanabe, Y.; Ishiguro, A. & Uchikawa, Y. (1999). Decentralized behavior arbitration mechanism for autonomous mobile robots using immune network. In: *Artificial Immune Systems and Their Applications*, Dasgupta, D., (Ed.), pp.187-209, Springer-Verlag, Berlin

Evolutionary Distributed Control of a Biologically Inspired Modular Robot

Sunil Pranit Lal and Koji Yamada
University of the Ryukyus
Japan

1. Introduction

Arguably the innovative problem solving abilities is one of the cornerstones for ensuring the survival of the homo sapien species in the game of evolution. Throughout history, when faced with challenges it was not uncommon for mankind to turn to nature for answers. In the modern day, problem solving utilizing techniques harnessed from nature has become a niche of the computational intelligence field. There are quite a number of classical contributions in this respect, which include artificial neural networks (ANN), genetic algorithm (GA), ant colony optimization (ACO), cellular automata (CA) and artificial immune system (AIS).

In the spirit of drawing inspiration from nature, our laboratory developed a modular robot modelled after a marine dwelling organism called the brittle star. The robot consists of independent modules with each module incorporating an onboard microcontroller for governing the behaviour of the module, actuator for inducing motion, and touch sensors for feeling the environment. Robot of such nature can be useful in search and rescue operations; for instance during earthquake the robot can be deployed to seek for survivors trapped under collapsed buildings which would otherwise be hazardous for human rescuers to reach.

Before novel applications of the robot can be envisioned the fundamental issues of motion control needs to be addressed. While the notion of studying the motion characteristics of the brittle star and incorporating it into the robot is intuitive and insightful, it is nonetheless quite impractical. The reason for this stems from the fact that the range of motion of the highly agile arms of brittle star in an aqueous environment overwhelmingly surpasses the two-degree of freedom legs of the landlocked robot. Hence we turn to nature once again, and attempt to draw from the evolutionary phenomena as a driving force to evolve the robot to move in its environment pretty much the same way the biological organisms have been shaped over billions of years by adapting to their environments.

Evolutionary algorithms are computational models, which capture the essence of evolution. Developed by John Holland in early 1970s (Holland, 1975), genetic algorithm (GA) is one of the widely adopted evolutionary algorithms. Inspired by biological adaptations, genetic algorithm is essentially a search technique used extensively to solve optimization related problems (Goldberg, 1989). The algorithm involves representation of candidate solutions to a problem using chromosomes also known as individuals. The initial randomly generated population of individuals is successively transformed based on their fitness by applying genetic operators

such as selection, crossover and mutation. The selection process screens the individuals such that the fitter individuals have higher probability of making it through to the next generation. The crossover operation essentially emulates the intraspecies mating by exchanging and combining genes from selected parent chromosomes to produce offsprings, with the hope that they may have better fitness than the parents. Mutation is a way to introduce diversity within the population, thus enabling better exploration of the search space. Based on the survival of the fittest, it is anticipated that with each passing generation the fitness of the individuals improves thus providing near optimum solution to the problem at hand.

In nature, organisms evolve to better adapt to their environment, which also consists of organisms from other species. Thus the interaction between the species also plays a role in shaping the evolution of organism. Co-evolutionary algorithm (Koza, 1991) is a computational model where two or more populations simultaneously evolve in a manner such that the fitness of an individual in a population is influenced by individuals from other populations. A classical example of this is the interaction between host and parasite species, whereby the host develops mechanisms to fend off the parasite, meanwhile the parasite finds ways to counter the defence mechanism of the host, thus setting off an arms race. This kind of external pressure, forces the populations to adaptively evolve thus avoiding suboptimal solutions to a large extent.

Using evolutionary approach to derive motion characteristics of the robot requires a control model. In this chapter we will discuss two streams of control model namely cellular automata based control and neural network based control model.

In the cellular automata based control (Lal et al., 2006) the individual modules of the robot are modeled as cells in the cellular automata lattice. Based on this we incrementally developed control models for the brittle star robot leveraging off genetic algorithm and co-evolutionary algorithm to evolve suitable rules for each of the models.

A shortcoming of the cellular automata based approach was that the evolved rules were tightly coupled with the initial configuration of the cell lattice. To address this, the neural network inspired control model (Lal et al., 2007) was developed. In this framework each leg of the robot is modeled as a neural network with the modules representative of neurons interconnected via synaptic weights. Motion is achieved by propagating phase information from the modules closest to the main body to the remainder of the modules in the leg via the synaptic weights. Similar to the cellular automata based model, near optimal control parameters were evolved using genetic algorithm.

2. The Brittle Star: From Biological to Binary

Brittle stars [*Ophiurida*] (Hyman, 1955) are echinoderms found in most of the marine ecosystems around the world. The physical structure (Fig. 1) of the brittle star consists of five slender and flexible arms in a radial symmetry attached the central disc shaped body, which houses all of the internal organs. An internal skeleton of calcium carbonate plates referred to as *vertebral ossicles* supports the arms. These are linked by ball and socket joints, and moved by the surrounding muscle.

The brittle star moves by wriggling its agile arms to produce snake-like or rowing movement. Being able to crawling through small cracks, or being able to move even when missing an arm or two, lost perhaps when evading a predator, or being able to regenerate the lost arm or segments of it, makes the brittle star a true star in the arena of adaptive systems. This was indeed the inspiration behind developing the brittle star robot.



Figure 1. The brittle star

2.1 The Brittle Star Robot

The brittle star robot (Fig. 2a) considered herein has a modular architecture consisting of modules as shown in Fig. 2b. Each module incorporates an onboard micro controller (BASIC Stamp 2sx), actuator (RC Servo Futaba S5301) and two touch sensors. In its current setup the actual robot hardware has five legs with six modules per leg.

The robot design, inspired by the brittle star has the following characteristics:

- The robot has capability to move in all directions with a body shape consisting of five arms radiating from the centre.
- Bone tissue of unit structure is built by connecting homogeneous modules. This allows redundancy, decentralization of control program and simplification of repair to the robot.
- Modules of one degree of freedom are connected alternately in horizontal and vertical orientation (Fig. 3). In this way, a set of adjacent modules has two degree of freedom joint.

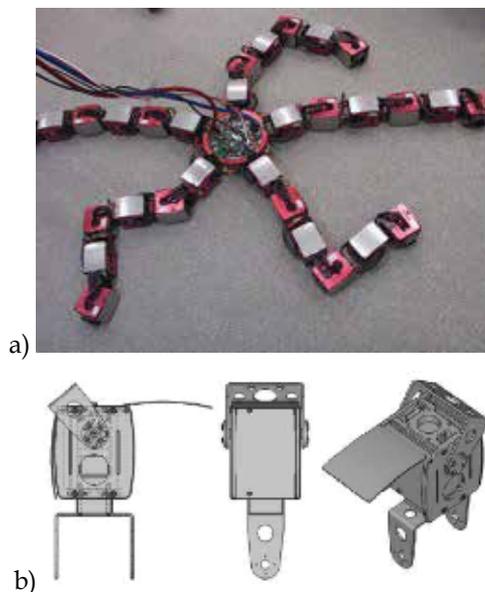


Figure 2. a) The brittle star robot b) Individual module connected to make up the leg



Figure 3. Exposed view of adjacent connected modules

2.2 Motion Control Problem Formulation

Moving the robot requires coordinated movement of the individual modules. Initially this was achieved through trial and error process (Takashi, 2005). Later genetic algorithm was used to better the process of finding near optimal motion pattern (Futenma et al., 2005) Though the approaches adopted in the past produced desired motion characteristics in the robot, the underlying concern was that should there be a failure of some modules the overall mobility of the robot would get compromised. To this effect, we have embarked on exploring strategies for evolving emergent global behaviour of the robot, such as walking in a straight line, from the combined action of all the functioning modules in the a given configuration.

One way to visualize the module is to consider it as a state machine. Since the motion of any given module is basically rotation between an allowable range, the state of j^{th} module in the i^{th} leg at time, t is given by (1)

$$S_{ij}^t \in \{ \theta : \theta_{min} \leq \theta \leq \theta_{max} \} \quad (1)$$

This makes the robot a collection of state machines, or to put it differently, a state machine of state machines. The state of the robot with n legs and m modules per leg at time, t is thus given by (2)

$$R^t = \{ S_{ij}^t : 0 \leq i < n, 0 \leq j < m \} \quad (2)$$

Therefore the problem of motion control of the robot becomes the task of finding optimal sequence (O_T) of state transitions, which would transform the robot producing desired locomotion.

$$O_T = R^0, R^1, R^2, \dots, R^T \quad (3)$$

It should not take much imagination to realize the sheer magnitude of the search space that needs to be explored to find near optimal solution, and thus the need for evolutionary computational approach.

2.3 Robot Simulation

While it is plausible to evolve motion controller on the physical robot itself, it is nonetheless highly impractical as doing so might damage the robot in evaluating the fitness of unknown

controllers, not to mention such an approach would be time consuming and unnecessarily cause wear and tear to the robot.

Simulated model (Fig. 4) of the brittle star robot was developed using Open Dynamics Engine (ODE version 0.9), which is an open source, high performance library for simulating rigid body dynamics (Smith, 2006). In developing the robotic simulation it is important to realize that the simulated robot is an approximation of the real robot, and as such choice of simulation parameters play a crucial role in determining if and how well the evolved control parameters can be implemented on the real robot.

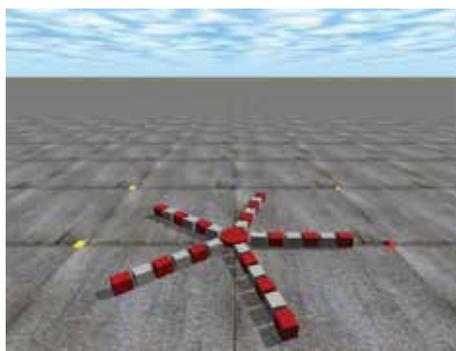


Figure 4. Simulated brittle star robot using Open Dynamics Engine

In our previous studies (Lal et al., 2006; Lal et al., 2007) we used torque as a parameter to control the rotation of the modules, which was simple to implement in ODE simulation. Later we realized that torque, as a parameter had no grounding on the physical robot because the control of the servomotor through the microcontroller involves varying pulse width to control the angle. Thus we have revamped our simulation model to centre on phase angle as the primary control parameter for the modules. It is worth mentioning that the while the control models presented in the following sections have largely remained similar to our previous work, informed choice of parameters have made significant difference in the results.

One of the parameters which influences all the control models presented in the following sections deals with the choice of number of states per module. The angular range $[-\pi/3, \pi/3]$ of the module was discretized into 16 states (0, 1, 2, ..., 15) which provided fairly granular control for the control models.

3. Cellular Automata Based Motion Control

3.1 Overview

Historically cellular automata (CA) emerged around 1940's (Wolfram, 2002) as a quest to develop computational mechanisms to resemble information processing systems in nature. John von Neumann's pioneering work in developing models for self-replicating automata (Neumann, 1966) set the stage for the future growth of this field.

Cellular automata consist of a lattice of identical finite state machines (cells) whose state changes is governed by some common transition rule. The next state of a cell at time, $t+1$ is determined from the current state of the cell and its neighboring cells at time t . Even the simplest rules can result in emergence of interesting patterns over a period of time, as in the case of Conway's game of life which is perhaps the best known two-dimensional cellular

automaton used to model basic process in living systems. Crutchfield et al. (2003) treatment of evolving CA using GA makes for interesting reading.

In the field of robotics cellular automata has been used to control self-reconfigurable robots, which can autonomously change their shape to adapt to their environment (Bultler et al., 2001; Stoy, 2006). Furthermore, Behring et al. (2000) has demonstrated promising use of cellular automata as a means to perform path planning using real robot in an environment clustered with obstacles.

In our approach, we modelled the individual modules of the robot as cells in the cellular automata lattice. Based on this we incrementally developed three control models for the brittle star robot. Genetic algorithm was used to evolve suitable rules for each of the models.

3.2 Singular Transition Rule for Disjoint Leg Set

In our initial attempt, the robot (with $n = 5$ legs and $m = 6$ modules per leg) was modelled as a set of disconnected one-dimensional CA lattice representing the legs of the robot and with each cell representing a module. Furthermore we decided to derive a singular CA transition rule for all the modules in the robot irrespective of their connection topology. The rationale behind this was that since the rule would operate on a small neighborhood of modules, should any of the modules or legs fail, the overall mobility of the robot should not get compromised to a large extent. Furthermore, a single unifying transition rule would make the modules truly modular from both hardware and software point of view thus maintenance would be a ease in terms of replacement of modules.

Each cell in the CA framework is representative of a module and it has $k = 16$ possible states to represent the angle of rotation of the module. The neighborhood, η of a cell can encompass any number cells in its vicinity. The smaller the neighborhood, lesser the communication overhead per cell update. Moreover this also translates into fewer entries in the rule table. For this reason we choose the neighborhood of a cell to include only its adjacent cells, that is radius, $r = 1$ (Fig. 5).

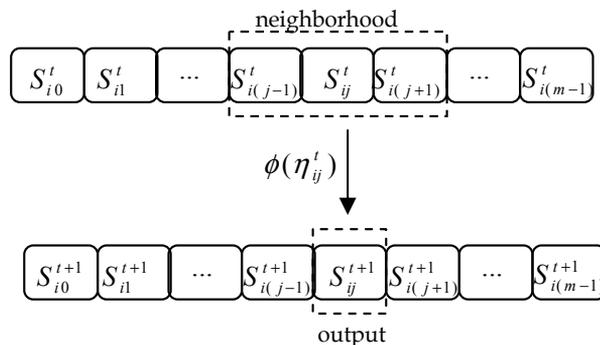


Figure 5. Lattice configuration showing states of the modules in the i^{th} leg of the robot. Radius $r=1$ groups 2 cells (modules) adjacent to the current cell to form a neighborhood, η which is applied as an input to the transition rule, $\phi(\eta)$. The resulting output is used to update the state of the current cell at time, $t+1$

To apply the same rule consistently on all the cells, we handle the cells on the edge of the lattice with the following boundary conditions for all $u \in \{1, 2, \dots, r\}$

$$S_{i(j-u)}^t = \begin{cases} S_{i(j-u+m)}^t & \text{if } j-u < 0 \\ S_{i(j-u)}^t & \text{otherwise} \end{cases} \tag{4}$$

$$S_{i(j+u)}^t = \begin{cases} S_{i(j+u-m)}^t & \text{if } j+u \geq m \\ S_{i(j+u)}^t & \text{otherwise} \end{cases} \tag{5}$$

As an illustration, for 0th leg with 6 modules and neighborhood radius of 1, the neighbors for the cell S_{00}^t are S_{05}^t and S_{01}^t . Likewise the neighbors for cell S_{05}^t are S_{04}^t and S_{00}^t .

3.2.1 Genetic Encoding

There are two key pieces of information, which needs to be encoded in the chromosome; the initial states of all the modules and the rule table for working out the state transitions. With 16 states per cell (4 bits), the first 120 bits of the chromosome is allocated for the initial state of the modules. For a neighborhood with a radius $r = 1$ the number of entries in the rule table (Table 1) is $16^{2r+1} = 4096$, thus 16384 bits for encoding the rule table.

The rule table is encoded in a chromosome as shown in Fig. 6. The output bits are listed in lexicographical order of neighborhood. Interestingly, even though the radius is minimum possible for cell-cell interaction to take place, the rather large number of states per cell has made the search space of possible transition rules phenomenal (2^{16504}). For this reason we did not consider increasing the radius.

Index	Neighborhood			Output
0	0000	0000	0000	0010
1	0000	0000	0001	0100
2	0000	0000	0010	0000
⋮				
4096	1111	1111	1111	1101

Table 1. A sample rule table for $(k,r) = (16,1)$

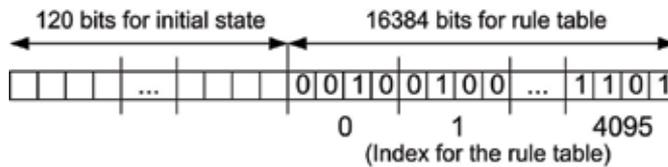


Figure 6. Representation of initial states and rule table in a chromosome

3.2.2 Fitness Function

The fitness of each chromosome is evaluated by first decoding the rule table it represents. The rule is then applied to the simulated model of the brittle star robot for successive state transitions (MAX_TRANS) thereby transforming it from initial position, (x_i, y_i) to the final position, (x_f, y_f) . Our primary focus is on forward locomotion of the robot, thus the fitness of the chromosome is proportional to the Euclidean distance covered by the robot.

$$F = \sqrt{(x_f - x_i)^2 + (y_f - y_i)^2} \quad (6)$$

3.2.3 Genetic Operators

Based on the fitness of the chromosomes in the population, GA operations; namely selection, crossover and mutation are applied to whole population. Firstly selection was performed using roulette wheel selection method. In addition the best individual in a generation is automatically carried over to the next generation as per the elite selection scheme. The selected pairs of chromosomes are crossed over using one-point crossover at randomly chosen locus with a crossover probability of P_C . Finally, mutation operation involving bit flips is applied with a probability of P_M to individual genes in chromosomes after the selection and crossover operations.

3.2.4 Simulation Results

The simulation was carried out over numerous trials using parameters shown in Table 2. In each trial, initial population of chromosomes of size (POP_SIZE) was randomly generated and GA was executed for a number of generations (MAX_GEN). For each generation, the fitness of all the chromosomes in the population is evaluated after which genetic operators are applied to the population to create the next generation.

In evaluating the fitness of a chromosome, first the initial state information encoded in the chromosome is used to initialize the simulated robot, and then the encoded transition rule is applied for a number of successive transitions (MAX_TRANS) thereby transforming the cell lattice. In each transition the object model in the ODE environment is updated accordingly. The final position of the robot at the end of MAX_TRANS transitions is used in fitness calculation above (6).

The evolution of transition rules across generations is captured in Fig. 7. As can be expected the average fitness of the population increased with passing generation.

Parameter	Value
PC	0.85
PM	0.15
POP_SIZE	25
MAX_TRANS	30
MAX_GEN	750

Table 2. Summary of simulation parameters

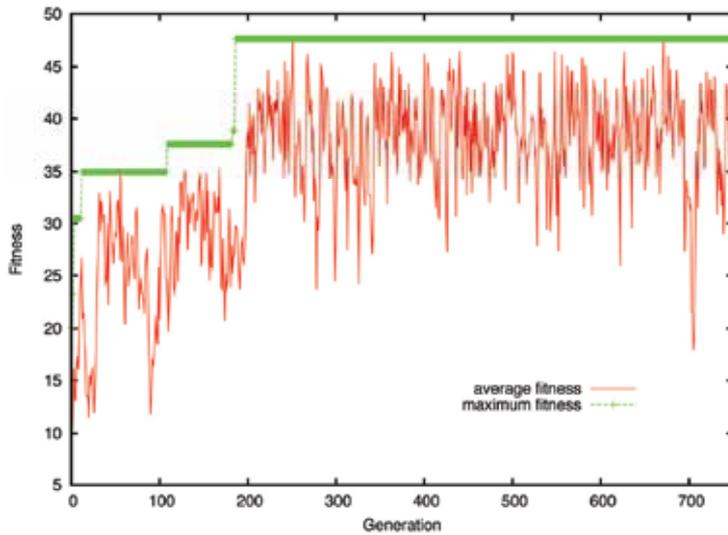


Figure 7. Graph of average and maximum fitness of population against generation for the disconnected one-dimensional CA model of the robot

3.3 Singular Transition Rule for Connected Leg Set

In this section, we extend the one-dimensional disconnected CA model by connecting modules closest to the central disc of the robot (Fig. 8) thus forming a two-dimensional structure. The notion of single transition rule for all the modules is adopted for reasons described in §3.2. Furthermore the radius describing the neighborhood of a cell is set to 1 (same as § 3.2).



Figure 8. The two-dimensional CA lattice. The arrows represent the interaction between a cell (possibly located somewhere in the middle or on the edge of the lattice) and its neighboring cells. Essentially the interaction is similar to the model in §3.2 except modules closest to central disc interact with modules in similar position located on the neighboring leg

The two neighbors ("right", R and "left", L) for a cell at position (i,j) in the $n \times m$ lattice is given as follows:

$$R = \begin{cases} S_{(i+1)j}^t & \text{if } j = 0 \text{ and } i+1 < n \\ S_{(i+1-n)j}^t & \text{if } j = 0 \text{ and } i+1 \geq n \\ S_{i(j+1-m)}^t & \text{if } j > 0 \text{ and } j+1 \geq m \\ S_{i(j+1)}^t & \text{otherwise} \end{cases} \quad (7)$$

$$L = \begin{cases} S_{(i-1)j}^t & \text{if } j = 0 \text{ and } i-1 \geq 0 \\ S_{(i-1+n)j}^t & \text{if } j = 0 \text{ and } i-1 < 0 \\ S_{i(j-1)}^t & \text{otherwise} \end{cases} \quad (8)$$

Using the same procedures and parameters described in §3.2, the simulations were carried out using the revised model, the results of which is shown in Fig. 9. Notably the fitness of the best chromosome discovered was comparable to that of the previous model.

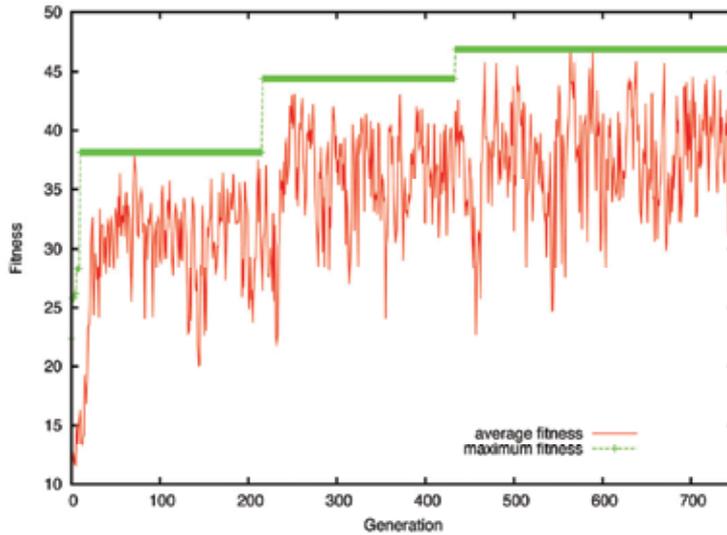


Figure 9. Graph of average and maximum fitness of population against generation for the two-dimensional CA model of the robot

3.4 Differential Transition Rule for Connected Leg Set

Building on the earlier models, in this section we present the final CA model, which represents the robot using a two-dimensional CA lattice as in the previous section, however there are two distinct transition rules for updating the cell states in the lattice. These rules are termed as control rule, CR and leg rule, LR . The position of a cell (i,j) in the lattice determines which of the two rules would be used to update its state and is given as follows.

$$Rule = \begin{cases} CR & \text{if } j = 0 \\ LR & \text{otherwise} \end{cases} \quad (9)$$

Essentially modules closest to the central disc (lead modules) are covered by control rule and modules located on other parts of the leg are taken care of by the leg rule.

3.4.1 Co-evolving the Rules

Given the differential nature of the rules we decided to employ co-evolutionary algorithm to discover the optimal rule set. Co-evolution can be competitive in a nature such as in (Rosin & Belew, 1995) where the population includes opponents trying to beat each other in a game of Tic-Tac-Toe, or it could be cooperative in nature such as in (Potter & Jong, 2000) where populations of match strings collaboratively contribute individual string to form a set of binary vectors which is to closely match a target set of binary vectors.

In our case a population of control rules and another population of leg rules is co-evolved cooperatively such that rules from both population need to be combined to control the robot. The genotype of individuals in the two population vary slightly in that the individuals in the population of control rule encode the initial state of lead modules whereas the leg rule individuals encode the initial state of the remaining (non-lead) modules. The encoding of state transition rules remain essentially the same as previous CA models. The following describes the pseudo code for the co-evolutionary algorithm.

Create random population of control rules \mathbf{P}_{CR} and leg rules \mathbf{P}_{LR}

LOOP UNTIL MAX_GEN

Get best control rule *//Rule with the highest fitness; for initial population*
//best rule is randomly chosen

LOOP UNTIL MAX_COEVOL_GENERATION

//Evaluate fitness of individual leg rules

LOOP FOR all individuals in \mathbf{P}_{LR}

- Transform lattice model using best control rule and current leg rule for MAX_TRANS
- Get final robot position
- Calculate fitness, $F = \sqrt{(x_f - x_i)^2 + (y_f - y_i)^2}$

END LOOP

Perform GA operations on \mathbf{P}_{LR}

END LOOP

Get best leg rule

LOOP UNTIL MAX_COEVOL_GENERATION

//Evaluate fitness of individual control rules

LOOP FOR all individuals in \mathbf{P}_{CR}

- Transform lattice model using best leg rule and current control rule for MAX_TRANS
- Get final robot position
- Calculate fitness, $F = \sqrt{(x_f - x_i)^2 + (y_f - y_i)^2}$

END LOOP

Perform GA operations on \mathbf{P}_{CR}

END LOOP

END LOOP

The simulations were carried out again using the same parameters as in §3.2.4, additionally MAX_COEVOL_GENERATION was set to 20. The results of search for best control and leg rule pair is presented in Fig. 10 & 11. As can be seen the fitness of the controller improved compared to the previous control models. This can be attributed to the fact that the interaction between the populations within the scope of the co-evolutionary algorithm creates greater diversity thus requiring greater adaptation on the part of the individuals in the populations. Consequently the stagnation of the search algorithm at local optima that was prevalent in the previous simulations was least encountered.

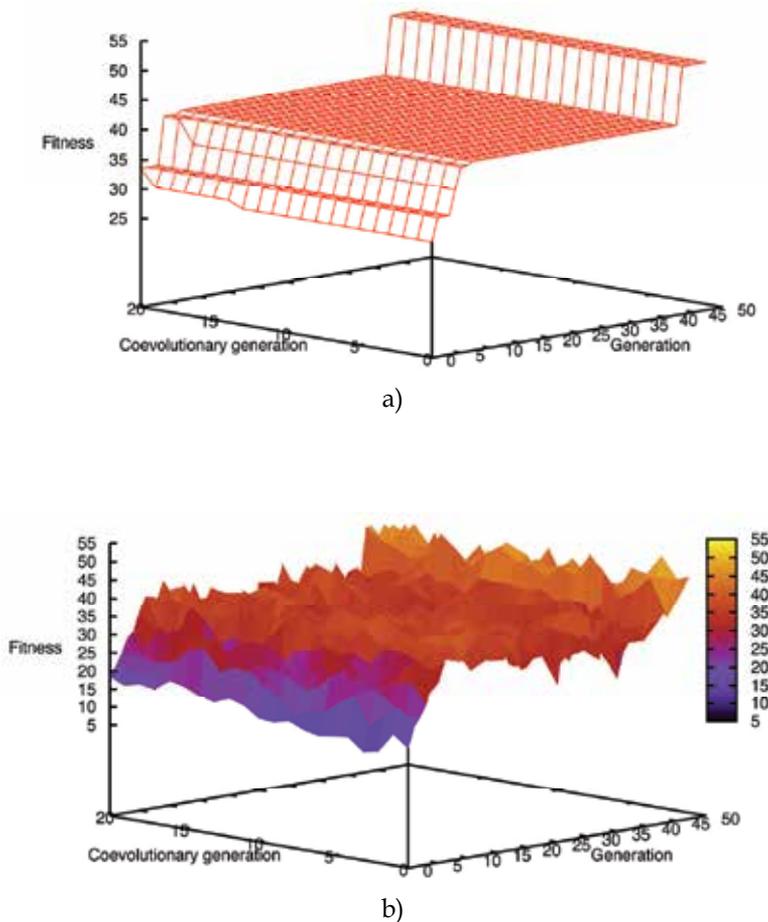


Figure 10. Graph of a) maximum fitness, b) average fitness, against generation for the control rule

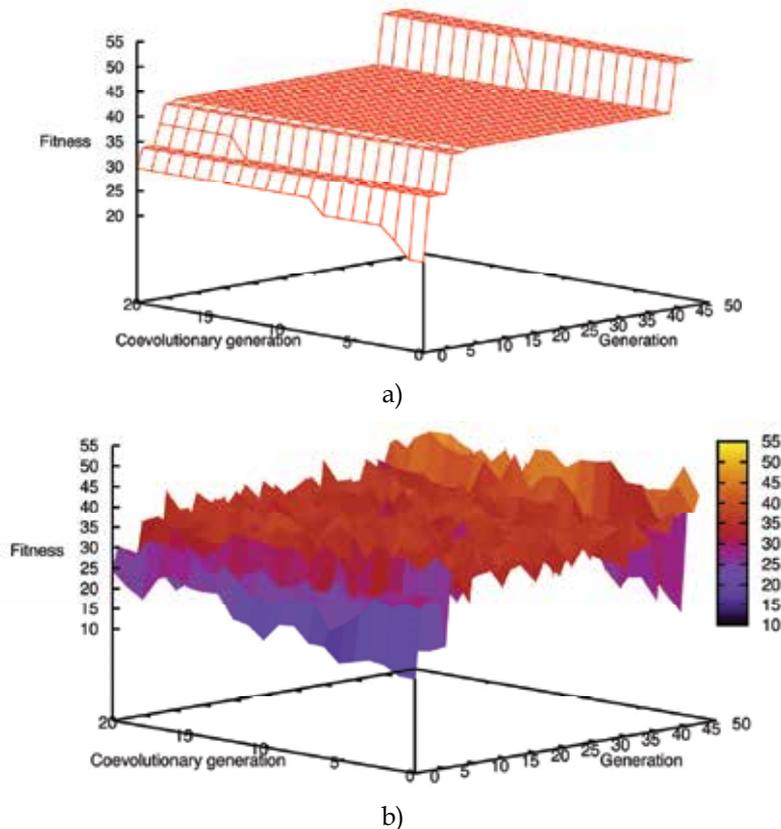


Figure 11. Graph of a) maximum fitness, b) average fitness, against generation for the leg rule

4. Neural network inspired Motion Control

While CA-based control architecture produced satisfactory motion some issues remained outstanding. In particular due to the inherent nature of cellular automata, the rules obtained were tightly coupled with the initial state of the lattice configuration. In other words if the initial phase angles of the modules is slightly changed then the resulting motion becomes incoherent. Moreover the shear size (2^{16504}) of the search space of possible transition rules made the task of learning computationally expensive. The model developed in the following sections tries to address these issues.

4.1 Overview

Inspired by the information processing ability of biological neurons, the modern day field of artificial neural networks, simply referred to as neural networks (NN) has its origins in 1943 with pioneering work by McCulloch and Pitts (1943) who developed computational model of a neuron. NN has been applied in multitude of areas namely pattern classification, function approximation, forecasting, optimization and control. While there are numerous neural network architectures in existence the fundamental computational model of the

neuron essentially remains the same. In a typical NN model the neurons are inter connected via synaptic weights. The neuron interacts by transmitting signals to other neurons connected to its output depending on the weighted sum of input stimulus to the neuron and the activation function. Learning takes place by adjusting the weights such that given an input, the output of the NN is as close as possible to the desired output. Interested readers are directed to (Jain et al., 1996) as good introductory reference material in neural networks. In the literature many notable contributions have been made in the field of robotics and control leveraging on GA and NN. Reil and Husbands (2002) successfully simulated bipedal straight-line walking using recurrent neural network whose parameters were evolved by GA. Porting genetically evolved neural network controller for a hexapod robot from simulation model to actual hardware was demonstrated by Gallagher et al. (1996). One of the conclusions reached by them was that neural network controller performed extremely well in real world in spite of the fact that inertia, noise and delays were not taken into account in the simulation. Hickey et al. (2002) developed a system called *creeper* featuring neural network controller for producing realistic animations of walking figures. The weights for the neural network were evolved using GA wherein the fitness of a chromosome encoding the weights was related to its performance in controlling the simulated walking figure. Application of neural network is not confined to controlling just single robotic agents as demonstrated by Lee (2003) in controlling behaviour of multiagent system of simulated robots in a predator and prey type environment. Similar to other research work described above, GA was used to evolve weights for the neural network behaviour controller.

4.2 The Control Model

Neural networks are good at dealing with system parameters whose relationships are not easily deducible. Inspired by this, we decided to model the interaction between the modules using the principles of NN. It is worth mentioning that the functionality of the model we developed though similar to conventional NN has subtle differences that are explained below.

Each of the leg is modelled as a fully connected neural network (Fig. 12) with the modules represented as neurons. The modules maintain state information about its current phase angle. Furthermore the modules are interconnected via binary weights to model inhibitory and excitatory stimulus between them. Formally, $w_{ab} \in \{0,1\}$, where w_{ab} represents weight between the connection from module a to b.

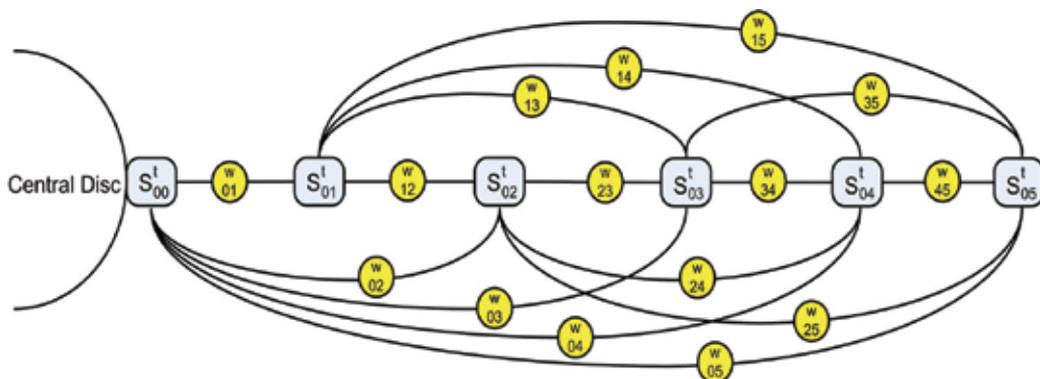


Figure 12. Conceptual framework for the neural inspired motion control architecture

Synchronized propagation of phase information through the network is used to update the current state of the modules. The modules closest to the central disc are termed as the lead modules. Starting from some initial state, the role of the lead module is to generate cyclic pattern in accordance with equation (10). Once the state of the lead module has been updated, the states of the remaining modules are then updated sequentially such that the module directly next to the lead module is updated first followed by the module next to it and so on.

$$S_{i0}^{t+1} = (S_{i0}^t + 1) \text{ mod } k \quad (10)$$

where k is the number of states per module

The state of any given non-lead module, S_{ij} in the leg is updated as follows. First the input stimulus from the modules closest to the central disc before it is summed.

$$X_{ij}^{t+1} = \sum_{k=0}^{j-1} \phi_{ik}^{t+1} w_{kj} \quad (11)$$

where ϕ_{ik}^{t+1} is the phase angle (in radians) associated with state S_{ik}^{t+1}

Next an activation function f_a is applied to the summed input to yield the phase angle, or in other words the equivalent state information of the module in question.

$$S_{ij}^{t+1} \Leftrightarrow \phi_{ij}^{t+1} = f_a(X_{ij}^{t+1}) \quad (12)$$

To put it intuitively, equations (11) and (12) allow a module to undergo valid state transition using latest state information of modules which underwent state transition just prior to it. In summary the state transition of the modules occurs sequentially within a discrete time step. Given the rotational nature of the modules we experimented using sinusoidal activation function (13) along with the traditional sigmoid activation function (14). It is worth mentioning that the parameters chosen for the activation function were done so as to keep the rotation of the modules in the allowable range $[-\pi/3, \pi/3]$ as shown in Fig. 13.

$$f_a(x) = \frac{\pi}{3} \sin(x) \quad (13)$$

$$f_a(x) = \frac{2\pi}{3(1+e^{-x})} - \frac{\pi}{3} \quad (14)$$

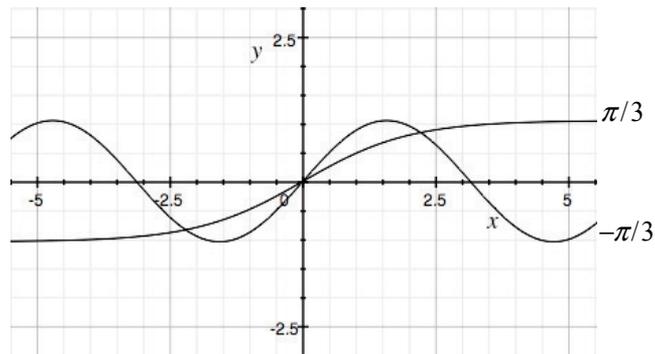


Figure 13. Graph of sinusoidal and sigmoid activation functions

4.3 Evolving Suitable Control Parameters

For the most part, GA framework was reused from §3 in determining near optimal initial states of the lead modules and the weight matrix for each of the legs. The only change to the GA framework required is encoding of the chromosome (Fig. 14). Notice unlike CA-based control model, the neural network control model requires only the initial state of the lead modules to function.

Compared to §3 the length of the chromosome has significantly decreased as it encodes 4 bits of initial state and 15 bits of weight matrix per leg. While real number could have been used for the weight matrix, in the interest of keeping search space manageable we decided just to use binary weights. Notably the search space (2^{95}) though significant is manageable in comparison with the search space for the CA-based model.

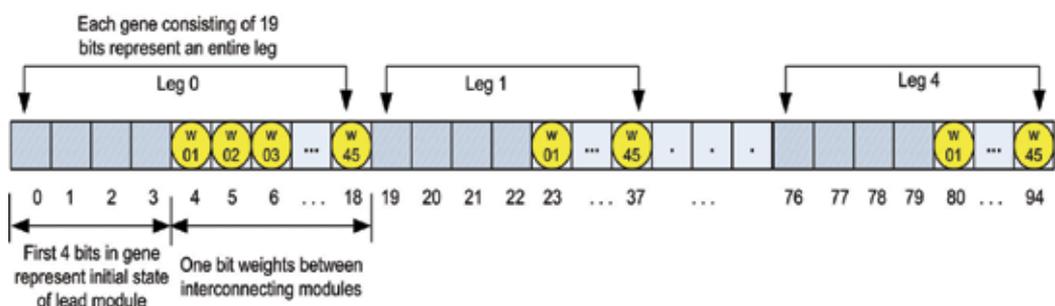


Figure 14. Genetic encoding of initial state and weight matrix

Using the similar procedures and parameters described in §3, the simulations were carried out using the revised model. The results obtained (Fig. 15, 16) indicated that the neural network model with sinusoidal activation function surpassed all the other models in terms of maximizing fitness.

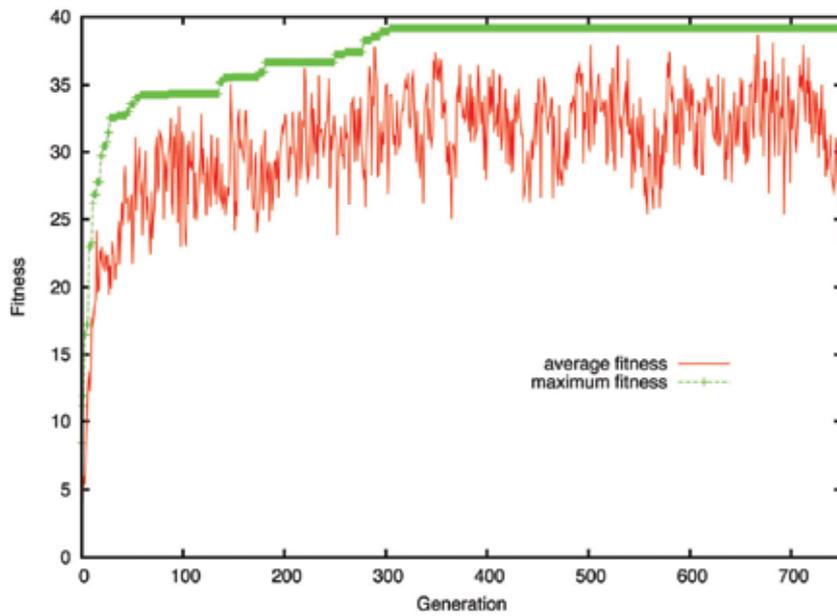


Figure 15. Graph of average fitness and maximum fitness of population against generation using sigmoid activation function

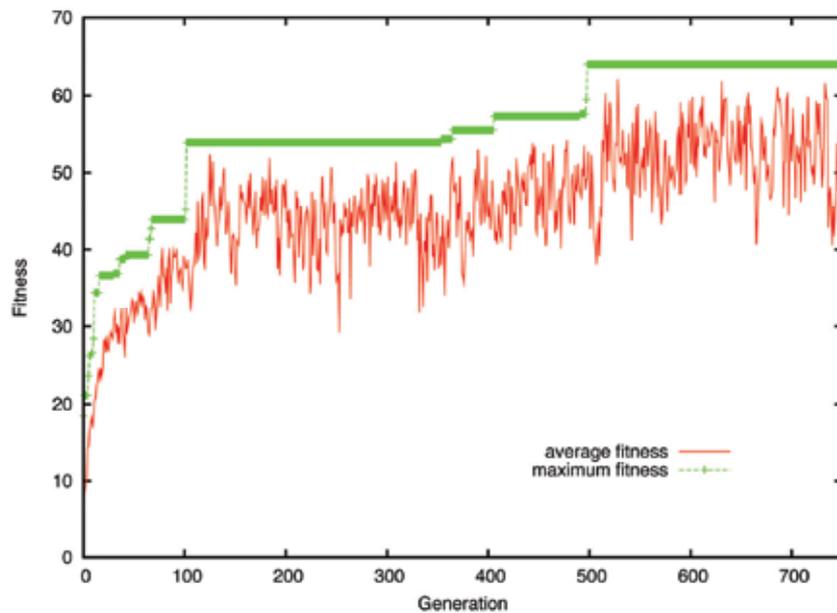


Figure 16. Graph of average fitness and maximum fitness of population against generation using sinusoidal activation function

5. Experimentation

In section we analyze the quality of the controllers evolved in terms of the motion characteristics produced and robustness of the control system in dealing with failure of modules. The following abbreviations have been adopted to refer to the controllers:

CA1d - One dimensional CA controller (§3.2)

CA2dSing - Two dimensional CA with singular transition rule (§3.3)

CA2dDiff - Two dimensional CA with differential transition rule (§3.4)

NNsig - Neural network controller with sigmoid activation function (§4)

NNsin - Neural network controller with sinusoidal activation function (§4)

5.1 Fault Tolerance

Evolving control parameters using GA is no doubt a time consuming process. Thus it would be highly desirable to have a robust control architecture, which can deal with module failures without having to be retrained.

In evaluating the robustness of the proposed models, module failure was simulated by configuring the module to be unresponsive to input stimulus thereby maintaining a fixed state. Simulation of the robot with a set of failed modules was carried out for 30 state transitions. The distance is then measured and used to calculate the degree of mobility (M_D) defined simply as:

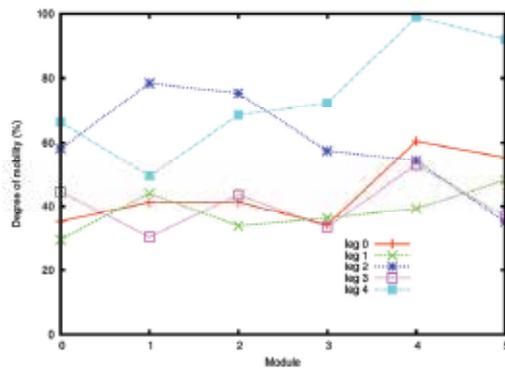
$$M_D = \frac{\text{distance traveled by robot with set of failed modules}}{\text{distance traveled by robot without failed modules}} \times 100\% \quad (15)$$

Failure was induced one by one in all the modules and the corresponding degree of mobility of the robot is depicted in Fig. 17. From the results it is apparent that the degree of mobility is greatly influenced by the position of the failed module, and the two streams of control model seem to handle failure differently. Overall the NN models performed better than CA models in the presence of module failure.

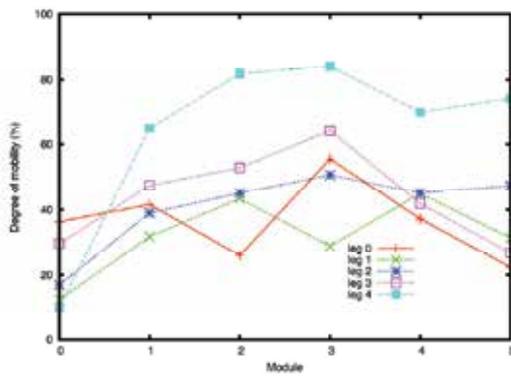
Within the CA models performance degradation was fairly distributed across the spectrum of individually failed modules. The two-dimensional models (CA2dSing & CA2dDiff) were more affected by the failure of the lead modules as it happens to be the point of connection between the legs. Overall CA2dDiff showed greater resilience compared to the other CA models.

Within the NN models the effects of failure of lead modules was noticeable. Needless to say the lead module is an important part of the control system as it provides the initial state information, and also the state transition of other modules is synchronized with the propagation of phase information from the lead module. Thus we can expect the failure of lead module would be the major contributor in hindering the overall mobility of the robot.

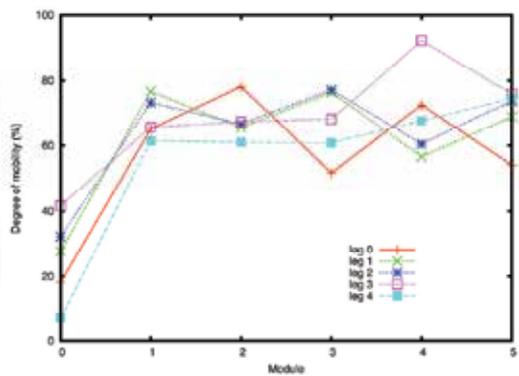
Finally, in the single module failure scenario, out of the 30 modules, 15 modules in NNsig and 8 modules in NNsin can fail with the robot still retaining 80% of its original mobility. Interestingly, for NNsin 6/30 (20%) of the modules can fail without compromising the mobility of the robot at all.



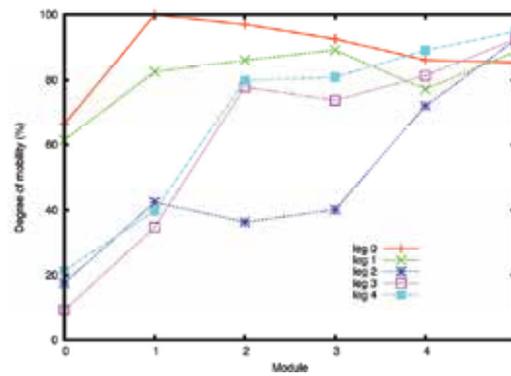
CA1d



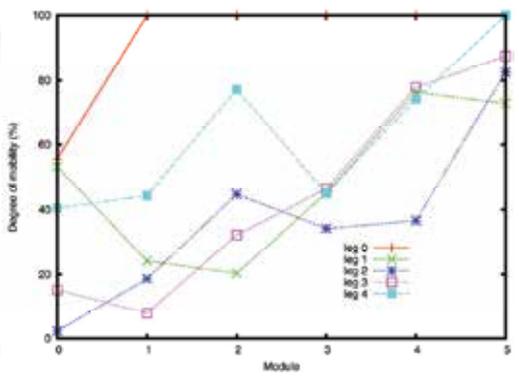
CA2dSing



CA2dDiff



NNsig



NNsin

Figure 17. Graph of degree of mobility against single module failure for all the models

5.2 Motion characteristics

Each of the controllers with best evolved parameters were run on the simulated robot for 30 state transitions, which is the same as the number of state transitions used by GA in evolving the controllers. The displacement of the central disc body of the robot from start to finish is depicted in Fig. 18. As can be seen the disc body of the robot did sway from side to side which is to be expected from robot of such nature. NNsig had the least amount of deviation on the straight-line path from start to finish. Though NNsig was the slowest of the lot, it nonetheless produced the smoothest motion.

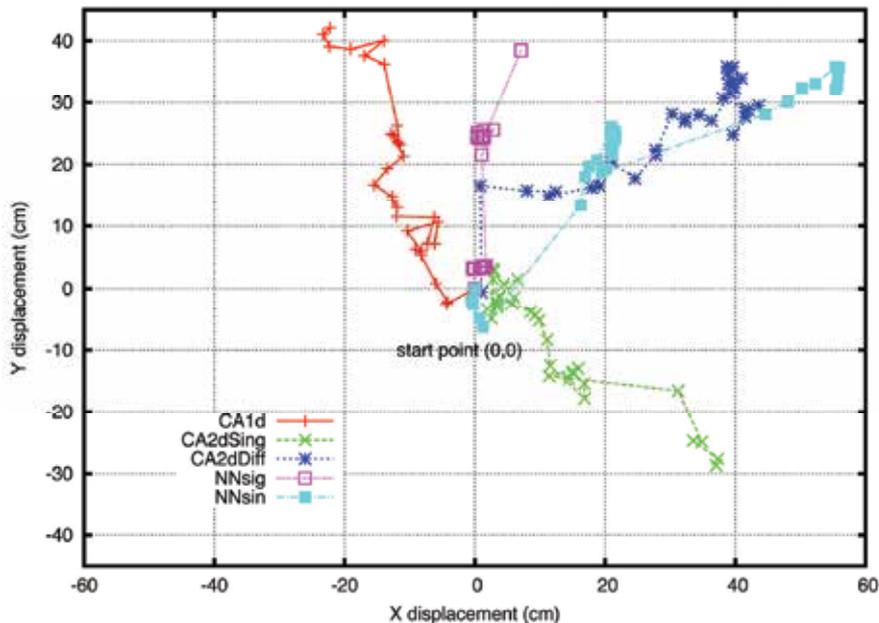


Figure 18. Displacement (cm) of the robot using each of the evolved strategies

An obvious observation is that the different controllers drove the robot in different directions, which would lead one to ponder as to how to get the robot to move in a desired direction. Well once the initial heading of the robot using a given controller has been determined, the robot can rotate about the centre of its disc body to the desired heading. An alternative to this can exploit the modular nature of the robot by swapping the control parameters between the legs and hence changing direction of motion.

In evaluating the fitness of a chromosome, the control parameters represented by the chromosome was applied to the simulated robot for $MAX_TRANS = 30$ state transitions. We are interested in knowing whether the results obtained can be extrapolated beyond this duration. Thus the motion of the robot using best control parameters discovered by GA was simulated beyond MAX_TRANS .

A clear demarcation in the behaviours of the CA based control models and NN based control models is captured in Fig. 19. The CA models peaked off around 30 state transitions after which their performance gradually degraded. Observations of the simulated robot using CA controller generally showed cyclic behaviour after exceeding MAX_TRANS . The NN controllers on the other hand made progress even after exceeding MAX_TRANS ,

though it must be pointed out that this rate became slower in the interval after MAX_TRANS.

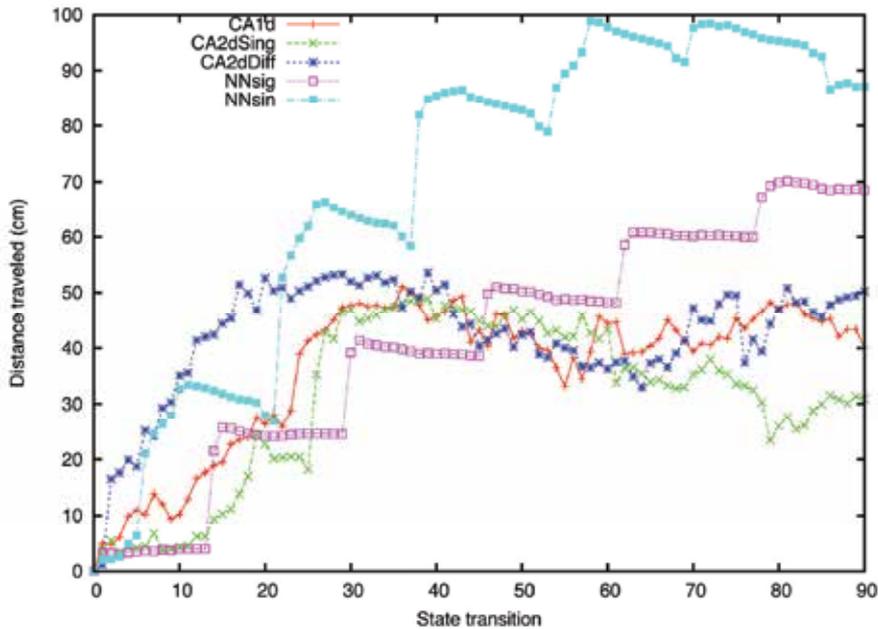


Figure 19. Graph of distance travelled against state transition using the best-evolved control parameters for each of the control models

The manner in which the controllers behave is intricately linked with the nature of evolutionary algorithm, which produced them. Genetic algorithm or for that matter any other evolutionary algorithm provide solution which is highly suited to a specific problem. In our case asking GA to evolve controllers to cover a maximum distance within 30 state transitions, will do exactly that. If we had used much larger number of state transitions, the evolved controllers except for covering greater distance, would display similar characteristics as the current controllers. One way to deal with this limitation is to let the robot move up till MAX_TRANS, reinitialize the state of the modules, and restart the state transitions from the beginning.

It is remarkable to see how the choice of activation function has influenced the strategy evolved by the neural network controller. Observations of state transition of individual module under the control of NNSig showed clustering of states, in other words given the current state of a module the probability that the next state will be the current state or the neighboring state was high. In contrast no such ascertains could be made about NNSin. The strategy employed by NNSin was quite unique compared to other controllers whose strategies resembled crawling motion. It involved coiling the legs, followed by lifting the disc body and then edging forward while uncoiling the legs and finally easing the disc body on to the ground as shown in Fig. 20.

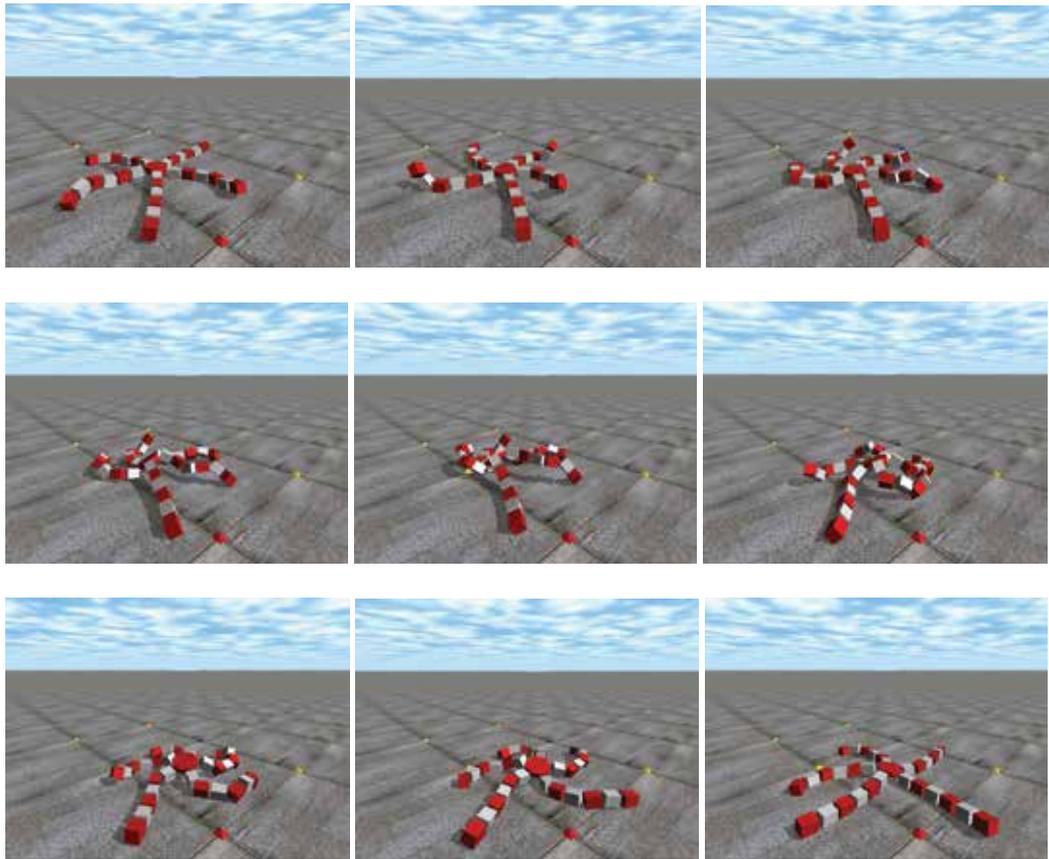


Figure 20. Snapshots of the robot's motion under the control of NNsin

6. Conclusion

The process of evolution by natural selection enables search for best solutions by adapting to the problem domain as time passes by. While we have utilized GA to find suitable parameters for our control models it is worth mentioning that from a global viewpoint the models in turn have evolved from trial and error means of motion control, to CA-based control architecture, to the neural inspired motion control model.

In light of the results, the CA2dDiff control model, whose parameters were derived using co-evolutionary approach, came out on top amongst the CA-based control models. However, the neural network based models showed better performance in terms of motion characteristics as well as exhibiting greater degree of resilience in overcoming scenarios involving failure of modules. The NN model with the sinusoidal activation function maximized the fitness function by covering the greatest distance, on the other hand the NN model with sigmoid activation function exhibited smoother motion characteristics.

Though the results are promising, this however does not imply the end of the road in the evolution of suitable controllers for the robot. There is still a lot that can be done to improve the robustness of the robot. One of the ideas is to leverage off the host-parasite analogy in coevolving a population of controllers with a population of test failure scenarios.

Evolutionary algorithms are goal driven. This makes them wonderful problem solving tools in that one has to describe the general framework of what the solution looks like and it does the rest by adaptively piecing the details. Being focused only on the end goal in terms of maximizing a fitness function is not always the best approach, as sometimes the route taken to achieving the goal matters more the end goal. In our case rather than focusing on maximizing the distance covered in a given number of state transitions, it is perhaps worthwhile exploring the nature of state transitions and seeking for specific states, which have the most influence in reaching the end goal. To this effect, we are considering markov model and reinforcement learning as future research options.

7. References

- Behring, C.; Bracho, M.; Castro, M. & Moreno, J. A. (2000). An Algorithm for Robot Path Planning with Cellular Automata, In : *ACRI-2000: Theoretical and Practical Issues on Cellular Automata*, pp. 11-19, Springer-Verlag, London
- Butler, Z.; Kotay, K.; Rus, D. & Tomita, K. (2001). Cellular Automata for Decentralized Control of Self-Reconfigurable Robots, *Proceedings of ICRA Workshop on Modular Self-Reconfigurable Robots*
- Crutchfield, J. P.; Mitchell, M. & Das, R. (2003). The Evolutionary Design of Collective Computation in Cellular Automata, In: *Evolutionary Dynamics-Exploring the Interplay of Selection, Neutrality, Accident, and Function*, Crutchfield J. P. & Schuster P.K. (Ed.), pp. 361-411, Oxford University Press, New York
- Futenma, N.; Yamada, K.; Endo, S. & Miyamoto, T. (2005). Acquisition of Forward Locomotion in Modular Robot, *Proceeding of the Artificial Neural Network in Engineering Conference*, pp. 91-95, St. Louis, Nov 2005, ASME Press, New York
- Gallagher, J. C.; Beer, R. D.; Espenschied, K. S. & Quinn, R.D. (1996). Application of evolved locomotion controllers to a hexapod robot, *Robotics and Autonomous Systems*, Vol. 19, pp. 95-103
- Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, Massachusetts
- Hickey, C.; Jacob, C. & Wyvill, B. (2002). Evolution of a Neural Network for Gait Animation. *Proceedings of Artificial Intelligence and Soft Computing*, Leung, H. (Ed.), ACTA Press, Calgary
- Holland, J. H. (1975). *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor
- Hyman, L. H. (1955). *The Invertebrates Volume IV: Echinodermata*, McGraw-Hill, New York
- Jain, A. K.; Mao, J. & Mohiuddin, K. (1996). Artificial Neural Networks: A Tutorial, *IEEE Computer*, Vol. 29, No. 3, pp. 31-44
- Koza, J. R. (1991). Genetic Evolution and Co-evolution of Computer Programs. In: *Artificial Life II, SFI Studies in the Science of Complexity Vol. X*, Langton, C. G.; Taylor, C.; Farmer, J. D. & Rasmussen, S. (Ed.), pp. 603-629, Addison-Wesley, Redwood City, CA
- Lal, S. P.; Yamada, K. & Endo S. (2006). Studies on motion control of a modular robot using cellular automata, In: *AI 2006: Advances in Artificial Intelligence - LNAI 4304*, Sattar, A. & Kang, B. H. (Ed.), pp. 689-698, Springer-Verlag, Berlin Heidelberg

- Lal, S. P.; Yamada, K. & Endo S. (2007). Evolving Motion Control for a Modular Robot, In : *Applications and Innovations in Intelligent Systems XV*, Ellis, R.; Allen, T. & Petridis, M. (Ed.), pp. 245-258, Springer-Verlag, London
- Lee, M. (2003). Evolution of behaviors in autonomous robot using artificial neural network and genetic algorithm, *Information Sciences*, Vol. 155, pp. 43-60
- McCulloch, W. S. & Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115-133
- Neumann, J. (1966). *Theory of Self-Reproducing Automata*, Burks, A. W. (Ed.), University of Illinois Press, Urbana, IL
- Potter, M. A. & Jong, K. A. D. (2000). Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents, *Evolutionary Computation*, Vol. 8, No. 1, pp. 1-29, MIT Press
- Reil, T. & Husbands, P. (2002). Evolution of central pattern generators for bipedal walking in a real-time physics environment, *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 2, pp. 159-168
- Rosin, C. D. & Belew, R. K. (1995). Methods for Competitive Co-evolution: Finding Opponents Worth Beating, *Proceedings of 6th International conference on Genetic Algorithms*, San Francisco, pp. 373-380
- Smith, R. (2006). *Open Dynamics Engine User Guide*. [Online]. <http://www.ode.org/>
- Stoy, K. (2006). Using Cellular Automata and Gradients to Control Self-reconfiguration, *Robotics and Autonomous Systems*, Vol. 54, pp. 135-141
- Takashi, M. (2005). *Studies on Forward Motion of Modular Robot*. MSc Dissertation, University of Ryukyus, Japan
- Wolfram, S. (2002). *A New Kind of Science*, Wolfram Media, Champaign, IL

Evolutionary Motion Design for Humanoid Robots

Toshihiko Yanase and Hitoshi Iba
*The University of Tokyo
Japan*

1. Introduction

The purpose of our research is to achieve a method of intuitive motion design that could be used by people who have no specialized knowledge of robotics in order to operate humanoid robots. Currently, as a result of the research conducted by Nishiwaki et al., stable walking of humanoid robots can be generated in real time (Nishiwaki et al., 2002). This method focuses primarily on walking, but it can easily be expanded to include any type of motion which is conducted on the same plane and with a level centre of mass. Nakaoka et al., using the method of walking motion generation devised by Nishiwaki, succeeded in capturing the motions of people dancing and enabling humanoid robots to mimic them (Nakaoka et al., 2003). In order to convert the motion expression obtained by capturing those motions into movement that could be realized by humanoid robots, they analyzed the primitive aspects of the leg motions. This made it possible to specify the primitive elements and the parameters, and to dynamically reproduce the leg movements performed in dancing in a stable manner. However, designing motion using the method devised by Nakaoka et al. requires a motion capturing system. It also requires people who can express the motions to be captured. Because of that, motion generation becomes a large-scale undertaking.

On the other hand, various methods have been proposed as intuitive methods for generating movements to be executed by humanoid figures in 3D-CG (3D computer graphics). These include the UT-Poser method proposed by Yamane and Nakamura. (Yamane & Nakamura, 2002), which uses pins to fix links and drags movable links, making it possible to generate any desired pose. There is also the Interactive Evolutionary Computation (IEC) (Takagi, 1998) method by Wakaki and Iba (Wakaki & Iba, 2002), in which any desired motion can be created simply by selecting a motion displayed on the screen. In motion design for humanoid robots, however, unlike in computer graphics, movements need to be generated that are actually feasible under the physical limitations imposed by the real world. For example, there are limits to the joint angles that can be expressed by humanoid robots, and when the robots come in contact with the ground, the soles of their feet have to be horizontal (we will explain the difficulties by examples in section 2.2). In order to apply motion generated using computer graphics with humanoid robots, this method has to be modified.

In our paper, the desired motion is designed using IEC as an intuitive method of motion design for humanoid robots and is aimed at people with no specialized knowledge of robotics and with no large-scale equipment. The aim is to achieve stable motions by following the desired ZMP (i.e., zero moment points). Because the motion generated via IEC is not necessarily the optimum motion, some form of optimization is necessary. Particularly in cases where the motion involves contact with an object that produces a reaction force, such as kicking a ball, optimization is necessary because of the reciprocity that occurs between the ball travel distance and the stability of the robot. For instance, Wolff and Nordin proposed an EC-based learning method in order to acquire stable biped walking for a simulated humanoid robot (Wolff & Nordin, 2003). However, this method requires a specialized gait controller manually developed to produce the initial population. On the other hand, in our approach, we can create the initial population from the motions generated via IEC.

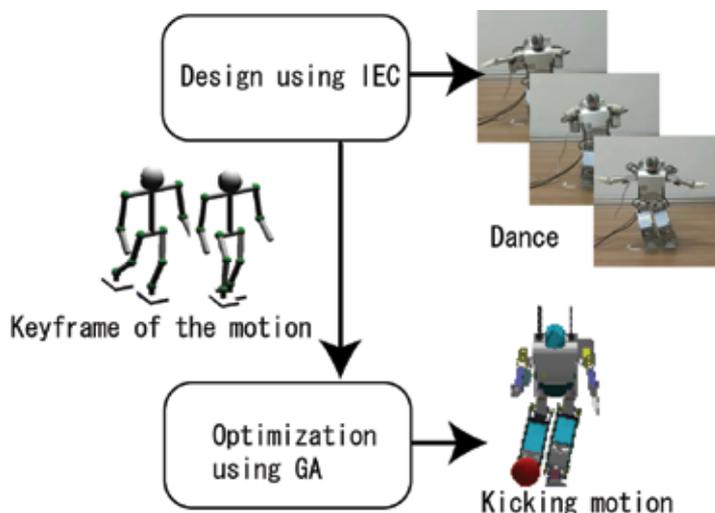


Figure 1. The flow of the experiment

In our research, we have used a dynamics simulator and carried out optimization using a genetic algorithm (GA) of the motions designed using IEC. Fig. 1 shows an overview of the experiment, which consists of two phases. In the first, IEC is used to evolve robot behaviours: users evaluate visually displayed robot motions that are generated with kinematic and stability constraints. In the second phase, GA is used to optimize the behaviour obtained from phase 1, by using the dynamics simulator.

This paper describes how successfully EC is applied to the generation of real motions for a humanoid robot. More precisely, we empirically show the following points by several experiments:

- IEC is effectively applied to designing the intuitive motions of a humanoid or a group of humanoids, e.g., kicking behaviour and cooperative dance.
- GA can be used to stabilize the motions generated by the above IEC-based method.
- The effectiveness of our approach is demonstrated by testing the generated motions with a real robot, i.e., HOAP-1.

The rest of this paper is organized as follows. In Section 2, we describe the experimental environment and the difficulties of generating humanoid's motions. IEC-based method is

proposed to avoid these difficulties. Their experimental results are shown in Section 3. Then, we explain how generated motions are optimized by means of GA in section 4. Thereafter, we discuss the results in Section 5 and give conclusion in Section 6.

2. Motion design for Humanoid Robot

2.1 Humanoid Robot and its Simulator

In our research, we have used the HOAP-1 (Humanoid for Open Architecture Platform) robot manufactured by Fujitsu Automation, as shown in Fig. 2. Motions were controlled by specifying the joint angles of the 20 joints of the entire body every 0.002 seconds. The characteristics of the HOAP-1 are noted below: (1) Height: 483 [mm], Weight: 5.9 [kg]. (2) The internal interface between the hardware and software is available for public use. (3) For movable parts, each leg had six degrees of freedom and each arm had four degrees of freedom, for a total of 20 degrees of freedom on the left and right sides. (4) The robot had the following functionalities: a joint angle sensor, a 3-axis acceleration sensor, a 3-axis gyro sensor and two foot load sensors.

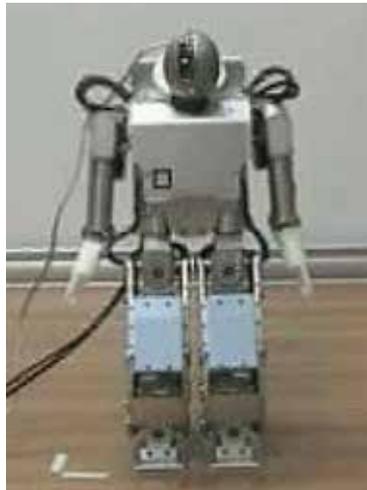


Figure 2. The HOAP-1

OpenHRP (Kanehiro et al., 2004) is used as the dynamics simulator. It is a software platform for humanoid robotics, and consists of a dynamics simulator, view (camera) simulator, motion controllers and motion planners of humanoid robots.

2.2 Difficulties of generating motions by using 3D-CG

When designing motions for a humanoid, it is essential to consider the contact with other objects and the external forces such as gravity so that the robots can move in a real environment. On the other hand, in case of 3D-CG humanoid figure, the motion design is more highly flexible, because the real-world constraints do not necessarily apply. Fig. 3 shows the typical design result of a kicking motion by means of 3D-CG, in which IEC was applied to generating keyframes of humanoid animations according to (Wakaki & Iba, 2002). We determined the motion of an avatar used in H-Anim in such a way that the amount of rotation of the joints could be obtained from the genes from IEC. In this design

example, we ignored gravity or forces of floor repulsion and fixed the position of the waist links. In the figures, (a) shows the keyframes of the designed kick motion. (b) gives the motion sequence between the first and second key frames, whereas (c) is the same motion sequence simulated considering the gravities and the repulsion force. As can be seen from the figures, the robot fell down when he raised his foot. This is because of the ignorance of external forces such as gravity or repulsion forces when designing motion by simulation. However, it is usually very difficult to include all the influences of these external forces beforehand for simple simulation. Therefore, the traditional 3D-CG technique has serious limitation to the application of designing robot motions in a real environment. The next section describes how IEC can generate stable motions by using ZMP calculation.

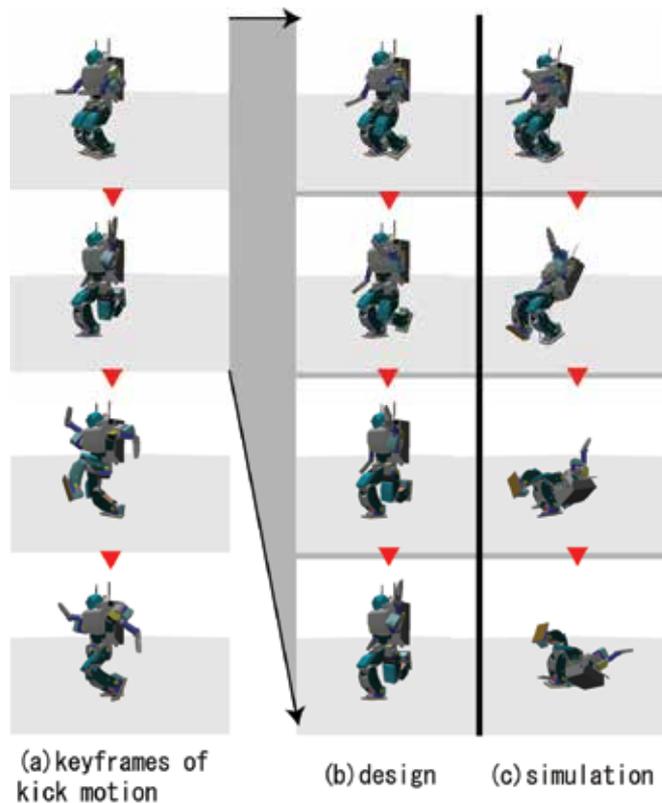


Figure 3. Applying 3D-CG method

3. Motion design using IEC

3.1 IEC-based motion design for a humanoid robot

Fig. 4 shows an overview of the motion design using IEC. The system provides the user with the motions generated using IEC. The design is created by the user looking at these motions and evaluating them. Everything that requires any specialized knowledge about humanoid robots, such as kinematics calculation, is done internally by the system, so the motions can be designed simply by having the user look at the screen and evaluate the motions.

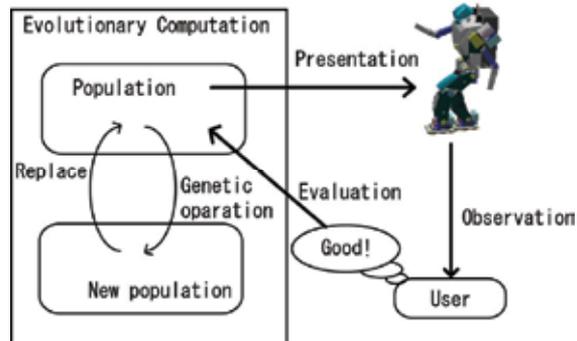


Figure 4. The IEC algorithm

3.2 Expressing motion

In our experiment, we have used the keyframe animation technique, which is often used with 3D-CG as a method of expressing motion. With keyframe animation, motion is expressed as the combination of a pose and a timeframe, and interpolation of the time between the poses is carried out to produce the animation. In (Wakaki & Iba, 2002), Wakaki et al. created the animation by creating the joint angle values for the entire body of the humanoid figure using interactive GA or interactive GP. With humanoid robots, if the joint angle values are set without taking the conditions of the support leg into consideration, there is very little possibility of generating an individual robot that will not fall down. With that in mind, we propose a method for specifying poses and an interpolation method, which together satisfy the conditions such that the humanoid robot will not fall down.

3.3 Pose Definition

Let us consider the process of assigning numeric values to poses in terms of keyframes. The weight of the robot has to be supported, and also important is the question of how to express the positions and attitudes of legs, because they might bump into or interfere with each other. If the poses likely to be taken by the humanoid robot are grouped based on the relationship of the feet to the floor, there are three cases to be considered: when both feet are in contact with the floor, and when one foot (right or left) is on the floor. If the position of the ankle of the support leg is determined, the state positions and attitudes that will keep the humanoid robot from falling over are limited, so the position and orientation of the ankle of the support leg should be set as the reference for poses of the entire body.

The position for the landing point of the support leg, as shown in Fig. 5, is decided based on the polar coordinates (r, θ) that use the support leg ankle position from the prior keyframe as a reference. The orientation is determined by the parameter ϕ . In order to prevent interference between the feet, the origin of the coordinates is offset from the ankle position of the support leg by the amount of the waist link, perpendicular to the orientation of the ankle. However, because the size of r that can be realized by means of θ is different, r is defined within a range solved by inverse kinematics. The ankle attitude is specified such that the sole of the foot is horizontal. This parameter is effective in cases when the support leg is changing, such as when shifting from one leg on the ground to both feet on the ground, or from the right foot on the ground to the left foot on the ground.

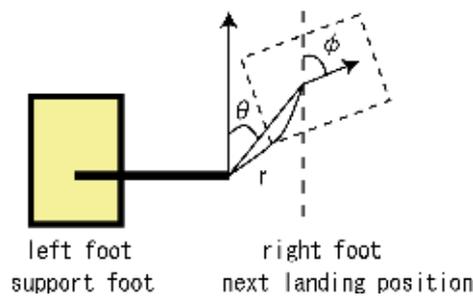


Figure 5. Specifying the ankle position of the support leg

The ankle of the swing leg is specified as shown in Fig. 6. The position of the ankle is specified using a cylindrical coordinate system (r, θ, h) . In this coordinate system, the origin is the position offset from the support leg ankle position by the amount of the waist link, perpendicular to the orientation of the ankle. Additionally, the ankle attitude is specified based on the roll angle, pitch angle and yaw angle. If the two feet are close together, or if the position of the swing leg ankle is near the floor, the feet may bump into each other in some cases, depending on the attitude of the ankle. In a case such as this, we restricted the attitude of the ankle to avoid collision. If both feet are in contact with the ground, this parameter is invalid.

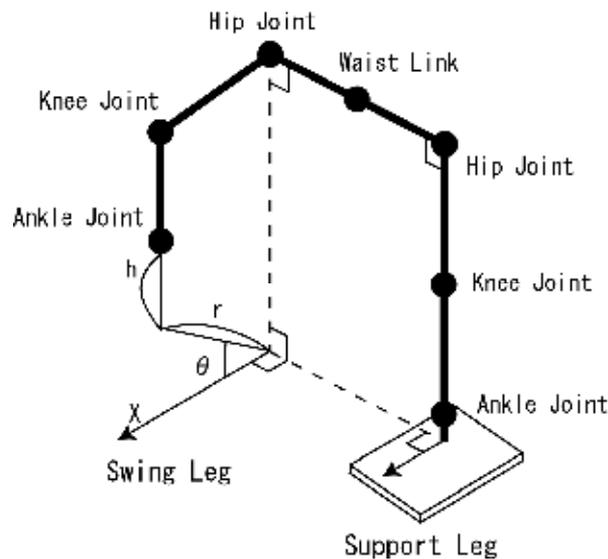


Figure 6. Specifying the position of the swing leg ankle

For the arms, which do not have to support the weight of the humanoid robot, we specified an arm joint angle with a total of eight degrees of freedom for the left and right arms together. For the waist link, we specified the height and the attitude.

The attitude was provided by the roll angle, pitch angle and yaw angle, using the support leg ankle link as a reference. When both feet were in contact with the ground, the ankle link of the right foot was used as a reference.

Table 1 shows a summary of the parameters used to decide the poses. Maximum and minimum values were determined for each parameter in advance, and these were handled by being normalized at $[0, 1]$. Because stabilization was the highest priority, the maximum and minimum values of the elements were set lower than the limit values.

	Element name	Degrees of freedom
(a)	Type of pose	1
(b)	Landing position and direction of support leg	3
(c)	Swing leg ankle-link position	3
(d)	Swing leg ankle-link attitude	3
(e)	Arm joint angle	8
(f)	Waist-link height	1
(g)	Waist-link attitude	3

Table 1. Pose parameters

3.4 Converting from a pose to motion

For the arms, which do not need to support the weight of the robot, the joint angles were smoothly interpolated using the natural cubic spline method. In converting the poses to motion, the interpolation has to be carried out under the condition that the calculation ZMP is not outside the actual support polygon, as explained below.

The centre of pressure of the floor reaction force in that state is called the ZMP (Vukobratovic & Stepanenko, 1972). A "support polygon", as shown in Fig. 7, can be defined as a convex closure that is the convex hull formed by the set of contact points of the robot and the floor. The left part of Fig. 7 shows the case in which both feet are on the floor, and the right part shows one foot in contact with the floor.

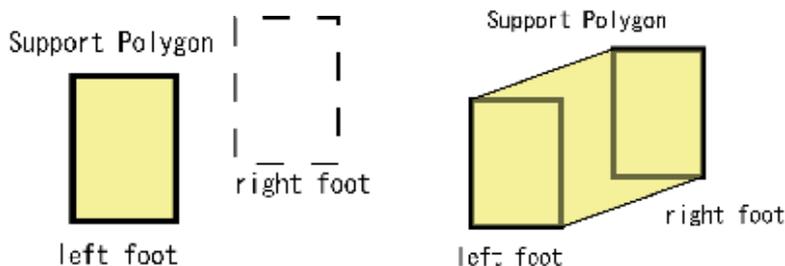


Figure 7. Support polygon

If the ZMP ends up being outside the actual support polygon, however, the robot will fall down. With that in mind, in order to prevent the robot from falling down in our experiment, we can correct the provided poses when interpolating them so that the ZMP resulting from the calculation of the physical model would not be outside the support polygon. The leg motions are derived according to the following sequence:

- Decide the landing position
- Derive the centre of mass position by the desired ZMP
- Calculate the support leg joint angles

First, the landing position of the support leg is decided. This derivation is based on (a) Type of pose and (b) Landing position and direction of the support leg. When the landing position is decided, the support polygon is also calculated. When one foot is on the floor

(Figure 8), the desired ZMP is taken as the centre of mass position of the support polygon. The waist-link height is the same as the one in the previous keyframe, and the waist-link attitude is upright. Next, the ankle position and attitude of the swing leg are decided based on (c) Swing leg ankle-link position and (d) Swing leg ankle-link attitude.

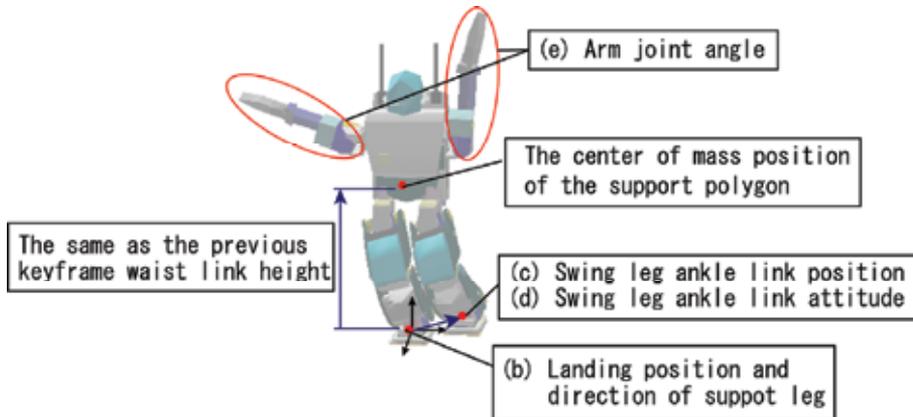


Figure 8. Converting from a pose to motion: when one foot is on the floor

When both feet are in contact with the floor (Fig. 9), the desired ZMP is decided based on (f) Waist-link height. Next, the waist-link attitude is derived from (g) Waist-link attitude.

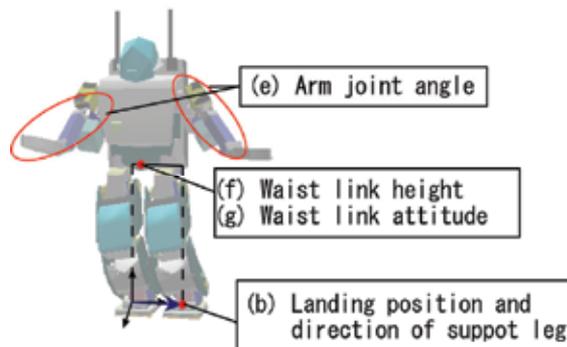


Figure 9. Converting from a pose to motion: when both feet are on the floor

3.5 Dynamic Balance

In order to determine the centre of mass trajectory that will satisfy the desired ZMP trajectory, as in the motion generation method used by Nakaoka et al. (Nakaoka et al., 2003), our method uses the fast generation method of motion pattern that follows the desired ZMP proposed by Nishiwaki et al. (Nishiwaki et al., 2002).

On a discrete system, supposing all the segments are restricted to be translated horizontally in the same distance, the following equation is acquired:

$$x_{zmp}(t_i) = \frac{-hx(t_{i+1}) + (2h + g\Delta t^2)x(t_i) - hx(t_{i-1}))}{g\Delta t^2}, \quad (1)$$

where x_{zmp} is a difference between a calculated ZMP and a desired ZMP, x is a translation distance of centre of mass to realize the desired ZMP, t_i is time at frame i , h is height of centre of mass, Δt is time per one frame. This equation is about x-axis, and similar equation applies to y-axis. This equation is expressed by information from 3 consecutive frames. These kinds of equations are solved as tridiagonal simultaneous linear equations. This method cannot figure out a result which completely follows the desired ZMP trajectory in one calculation because the constraint that all the segments translate parallel in the same distance is actually impossible. However, by iterating the calculation, a converged result is acquired. After that, the waist-link position is calculated from the centre of mass position. The support leg joint angles are decided using inverse kinematics so that the relationship between the waist-link position and ankle position is satisfied.

3.6 Setting the evolution calculation

When carrying out motion design using IEC, it is conceivable that the poses and times for all of the keyframes could be searched for at one time, but the large number of degrees of freedom in the joints of humanoid robots may make the search range too huge. For this reason, we progressively generate each pose and time by IEC, according to the following method.

	Element name	Degrees of freedom
(c)	Swing leg ankle-link position	3
(d)	Swing leg ankle-link attitude	3
(e)	Arm joint angle	8
(f)	Waist-link height	1
(g)	Waist-link attitude	3

Table 2. Elements of GTYPE: IEC

The real-valued GA is used to optimize the 18 parameters (shown in Table 2) of the pose of the final time. Thus, GTYPE consists of 18 real values. In most cases, the landing position can be determined from the travelling direction of the robot. Deciding the landing position by IEC prevents the convergence of the other parameters. We indicate the parameters of the landing position by using a simple GUI instead of IEC. The position (r, θ) can be specified by clicking in the predetermined feasible area, and the direction (ϕ) can be changed by dragging the footstep.

The motion calculation is carried out as follows:

1. A new keyframe is added to the motion, and the pose is randomly generated.
2. The poses are converted to the motion.
3. The motion is shown to the user.
4. The user evaluates the individuals. The design is terminated, if a sufficient pose is obtained.
5. The next-generation is created. Then, return to step 2.

Figure 10 shows a snapshot of our IEC-based motion design system. The user looks at the motions displayed on the screen and uses the slider under each pose to provide an evaluation value. In the displayed motions, the calculated joint angles have been played back using forward kinematics, and are not the result of a dynamics simulation.



Figure 10. Evaluation screen for IEC

3.7 Stability of generated motions

We provide a stability comparison between the proposed method and the 3D-CG method described in section 2.2. In this experiment, randomly generated motions were evaluated in the dynamics simulator. The motions were expressed by three keyframes. The first and third keyframes were fixed upright postures, and only the second keyframe was randomly generated. After interpolating the keyframes, the motions were evaluated by whether the robot fell down or not in the dynamics simulator.

Method	Number of success	Number of failures	Success rate (%)
3D-CG method	9	991	1
Proposed method: total	974	28	97
Proposed method: both feet	326	8	98
Proposed method: right foot	326	8	98
Proposed method: left foot	322	12	96

Table 3. Success rate of randomly generated motions

The experimental result is shown in Table 3. The success rate of the motion generated by 3D-CG method is very low. So if 3D-CG method is applied to motion design for a humanoid robot, all the individuals must be inspected using the dynamics simulator. Moreover, it is usually very difficult to evolve such highly-fatal individuals using ordinary Evolutionary Computation.

On the other hand, the proposed method generates stable motions with a much higher rate than the 3D-CG method. ZMP errors and interferences during the interpolation period between the keyframes were observed as causes of the failed motions for the proposed method. Both of them can be detected during the interpolation stage. Therefore, the proposed method enables the effective generation of stable motions.

3.8 Evolved behaviours of a humanoid robot

Figure 11 shows the simulation result for a kicking motion. The elements that were searched for were only the types of poses and the position of the right ankle, which was the swing leg. All the other parameters were fixed. Moving slowly, the robot was able to achieve a kicking motion without falling down, although the ball travelled only a short distance. With the kicking motion, the robot came in contact with the ball as well as with the floor, so there was offset between the ZMP that was obtained through the calculation using IEC and the actual ZMP. The stronger the reaction force from the ball, the larger the offset became.

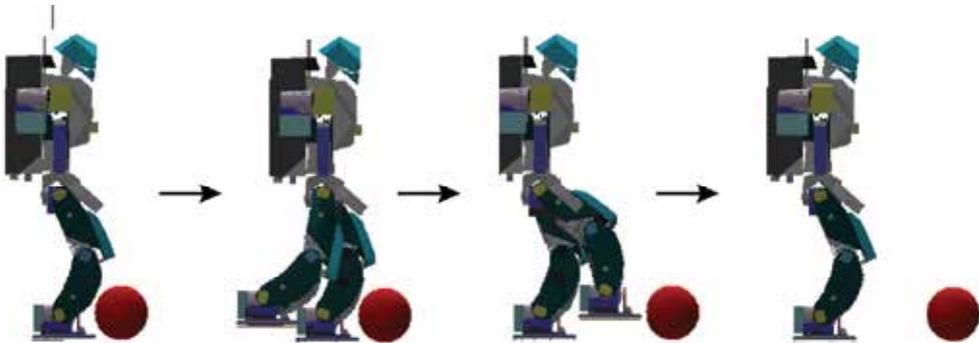


Figure 11. Example of motion design: Kicking

Another example is the cooperation dance of multiple robots. The dance of each robot was created with the above-mentioned method (see Figure 12). Although the robot moved slowly, the dance, which included the elements of a shift in the centre of mass and a tilting of the upper body, was carried out by the actual robots. With the simulator, the generated motion was stable without adding any particular changes. With the actual robots, however, when the dance was executed by the three HOAP-1 robots, one of the robots was unstable when lifting its feet. When the motions were modified so that the feet were not lifted as high, all three robots demonstrated the stable motion shown in Fig. 13. The learning process of cooperative behaviours among humanoid robots is described in details in (Inoue et al., 2004, 2007).

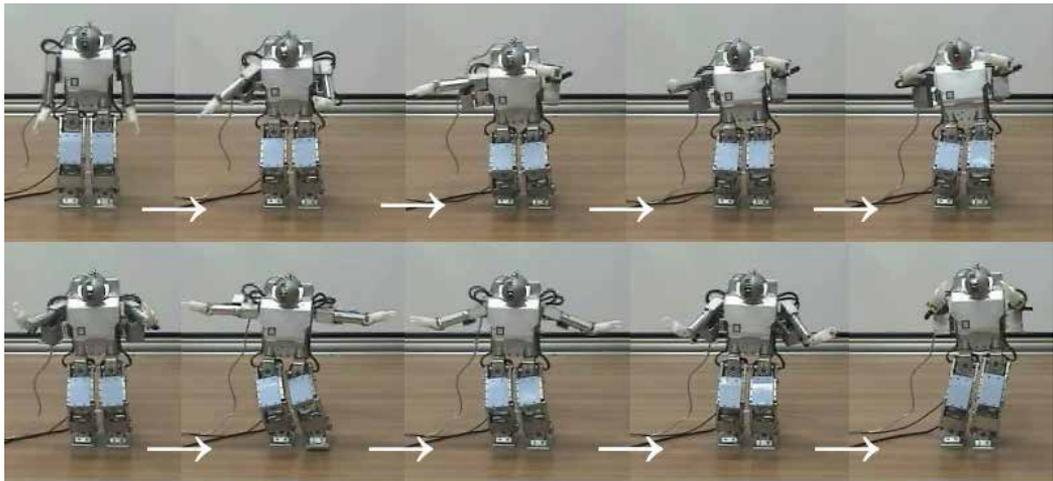


Figure 12. Example of motion design: A single dance

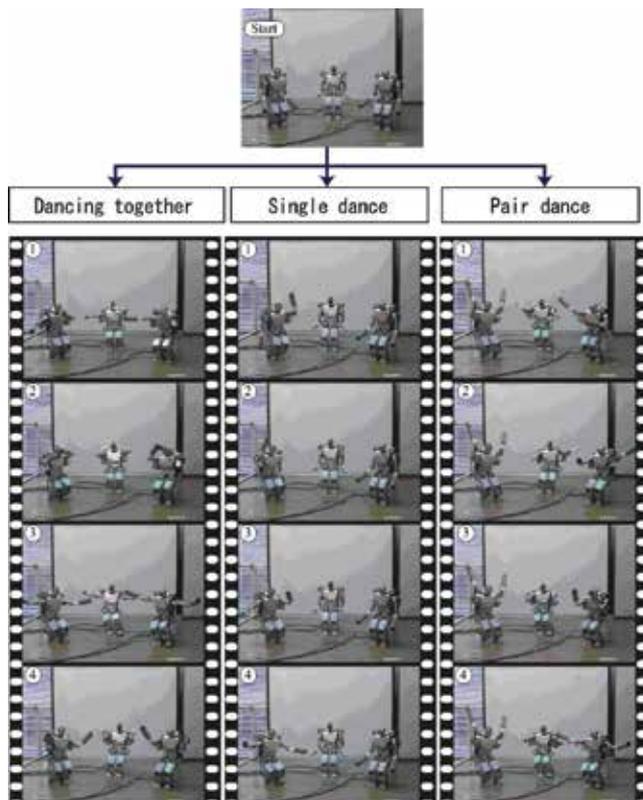


Figure 13. Example of motion design: Cooperative dance

4. Motion optimization using a GA

In the previous section, motions were generated through IEC according to the relation between the support foot and ZMP. However, generated motions were not necessarily stable. This is because the precondition for the stability, i.e., supporting polygon or external forces, is dynamically changing, especially when a robot is in contact with some object which is not a floor. In order to solve this difficulty, we employ real-valued GA for the sake of optimizing the generated motions in terms of the stability. This section describes two successful examples, i.e., optimizing a kicking motion and a sitting motion.

4.1 Optimizing a sitting motion

In order for a humanoid robot to sit on a box from a standing position, he has to move his gravity centre from his foot back to the contact place of the box and his waist link. Thus, it is very difficult to design by using only IEC.

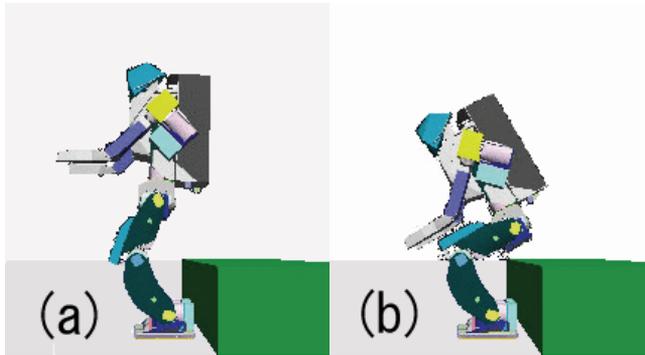


Figure 14. Sitting motion: Initial Pose, Final Pose

Suppose that two keyframes shown in Fig. 14 have been generated from IEC. Fig. 14 shows an initial standing position, whereas a robot finally bends his hip and knee joints at 90 degrees in Fig. 14.

We use GTYPE encoding the following items:

- the angles of the hip joint, the knee joint and the ankle joint for the intermediate position and the final position
- the time step for the intermediate position
- the time step for the final position

The fitness value is derived in the following way:

$$fitness = \exp(-\sin(\theta_{max})) + \exp(-V_{max}) + 2 \exp(-r), \quad (2)$$

where θ_{max} , V_{max} , r are defined as follows:

- Maximum lean of chest link during the motion (θ_{max})
- Maximum velocity of chest link during the motion (V_{max})
- Distance between waist link and surface of box (r)

The terms of $\exp(-\sin(\theta_{\max}))$ and $\exp(-V_{\max})$ are expression for evaluating the stability, whereas $\exp(-r)$ represents how successfully the task is achieved. In the above definition, $\exp(-r)$ is multiplied by two for the purpose of equalizing the two evaluation criteria.

Fig. 15 shows the evolved motion in a typical run. As can be seen from the third and fourth snapshots, the robot turns his angle joint and bends his body so that he can sit while keeping his gravity centre within the support polygon. The fourth and fifth snapshots show the contact of the corner of the waist parts with the box. In the fifth and sixth snapshots, the robot moves its gravity centre to the box while turning round the corner of the waist parts, as a result of which the robot can successfully achieve the sitting task.

Figure 16 plots the waist position in x and z coordinates for the best evolved individual and the linear interpolation method. The linear interpolation is commonly used in robotics for the sake of interpolating poses, e.g., Sony SDR-4X (Kuroki et al., 2003). Initial position corresponds to the upper right corner, whereas final position is on the lower left corner. Dots are plotted every 0.2 second so that the slower the moving velocity, the narrower the interval between the two dots.

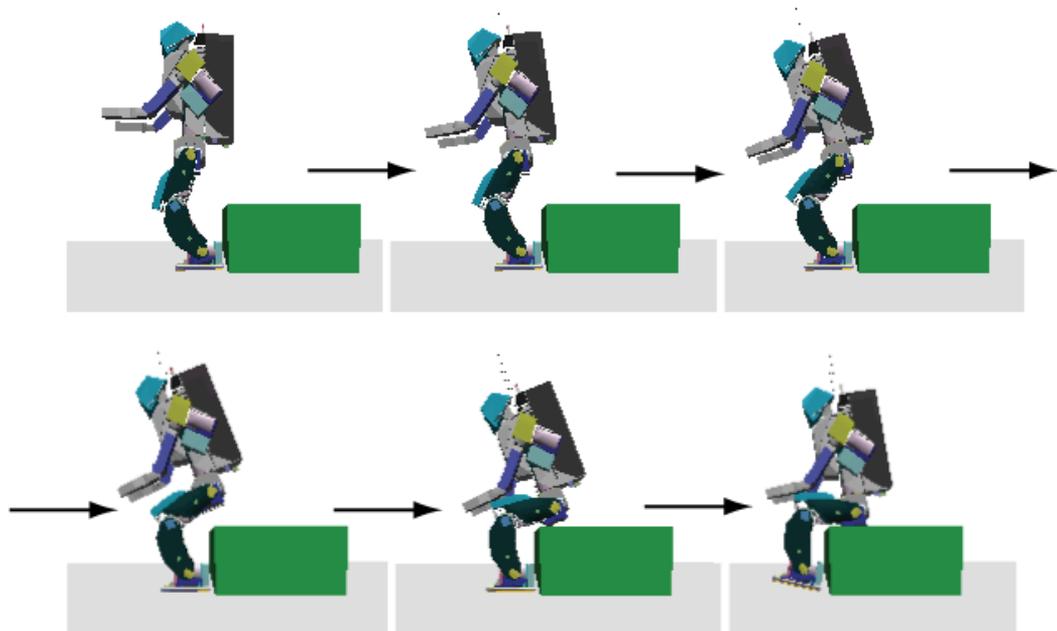


Figure 15. Sitting motion: best individual

Note that the slope of the best individual's curve is decreasing, which means that robot motions are changing gradually from vertically to horizontally. In the left corner, the slope is suddenly changed. This is the branching point when the robot moves its weight from the support polygon of its foot back to the box surface. The vertical distance of the waist link at that time was less than 5[mm] for the best individual, whereas it was about 20[mm] for the linear interpolation. The intervals between two dots are relatively wider for the linear interpolation, which means the motion velocity is faster. On the other hand, the narrow

intervals of the best individual reflect the slow motions, which can be observed in the fifth and sixth snapshots in Fig. 15.

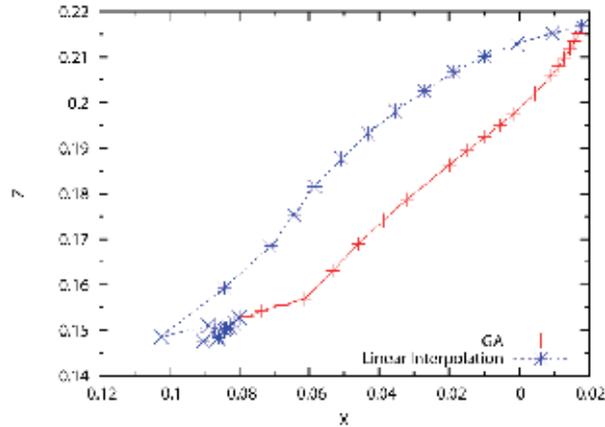


Figure 16. Sitting motion: Position of waist link

4.2 Optimizing a kicking motion

The kicking motion created using IEC in the previous section was carried out slowly, so that the ball did not travel a great distance.

In order to increase the distance travelled by the ball, the speed at which the tip of the foot is moving at the instant it contacts the ball has to be increased. However, the following elements, which reduce the stability, also increase as the speed of the tip of the foot increases:

- The reaction force from the ball
- The raising moment of the foot

Thus, we try to increase the distance travelled by the ball by using the real-valued GA to search for the poses in a keyframe and the pose times as explained below. The four keyframes of the kicking motion created using IEC, shown in Figure 11, were provided as the initial conditions. Three new keyframes were also added, in which the right foot was in the intermediate position of each pose, so searching was done with a total of seven keyframes. In this experiment, in order to obtain smooth motions, we have interpolated the intervals between keyframes using the natural cubic spline method.

Using real-valued GA, we searched for swinging of the arms in the forward and backward directions, raising of the right foot, and swinging of the upper body in the forward and backward directions. The GTYPE used in our experiment has the eight real elements shown in Table 3 for each keyframe. Because the operation targeted five keyframes, excluding the first and last keyframes, and the time of the last keyframe, a total of 41 real-value parameters were optimized.

For the position of the ankle link, global coordinates (x, y, z) were used as the GTYPE elements. Because of the usage of global coordinates, the positions specified for the ankle link are sometimes unlikely to be feasible, but in those cases a lethal gene results.

For each individual, a dynamics simulation was carried out using OpenHRP, and the results were used to evaluate the robot. The fitness value was provided using the following equation:

$$fitness = \exp(-V_{max}) + (1 - \exp(-r)), \quad (3)$$

where the maximum value for the chest-link velocity (V_{max}) and the ball travel distance (r) were used as the evaluation. V_{max} is the penalty in relation to the instability of the motion, and should be as small as possible. The fitness value for any robot that fell down was set to be 0.

Element name	Degrees of freedom
Time	1
Waist-link attitude	2
Arm joint angle	2
Ankle-link position	3

Table 4. Elements of GTYPE: kicking

Fig. 17 shows the fitness transition for the best individual with generations. A significant increase is observed in the distance travelled by the ball from the 10th to the 20th generation, with an accompanying increase in the fitness value. The stability decreased soon after that. Then, the stability subsequently recovered.

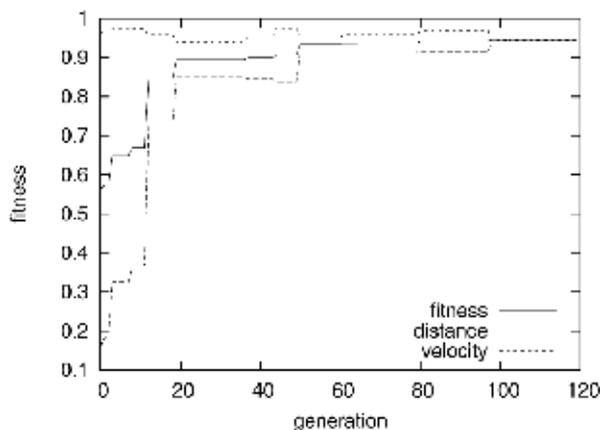


Figure 17. Fitness transition for the kicking motion

Fig. 18 shows the best individual at the initial generation (ball travel distance = 0.17 [m]). Fig. 19 shows the best individual after 120 generations (ball travel distance = 2.91 [m]). The numbers of Fig. 18 and Fig. 19 show the times (sec.) of keyframes.

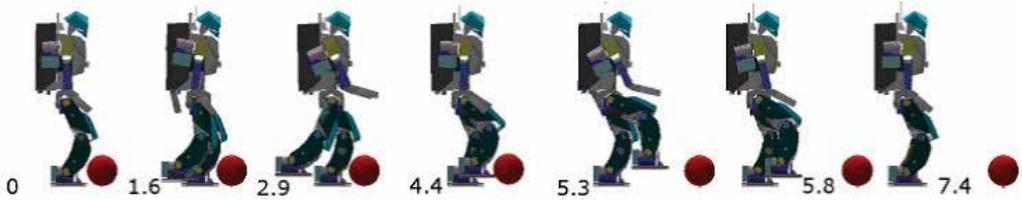


Figure 18. Best individual at the initial generation

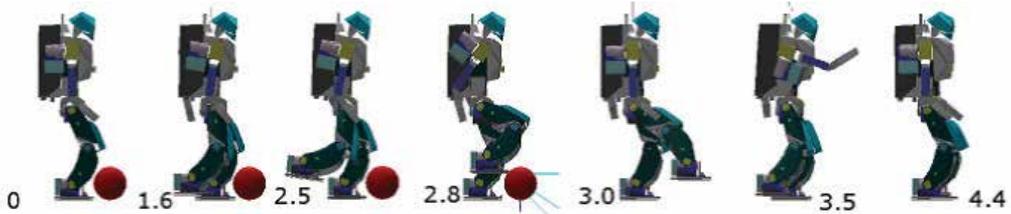


Figure 19. Best individual after 120 generations

The initial generation largely comprised motions without much variety that were performed at the same overall pace. The 120th generation, on the other hand, included varied motions that had the best individual slowly raising its foot upward from 0 seconds to 2.5 seconds and then slowly returning its foot to the floor from 3.0 seconds to 4.4 seconds, followed by rapid motions such as swinging its foot from 2.5 to 3.0 seconds. Because of this, the travel distance expanded from 0.19 [m] to 2.91 [m], or by approximately 17 times.

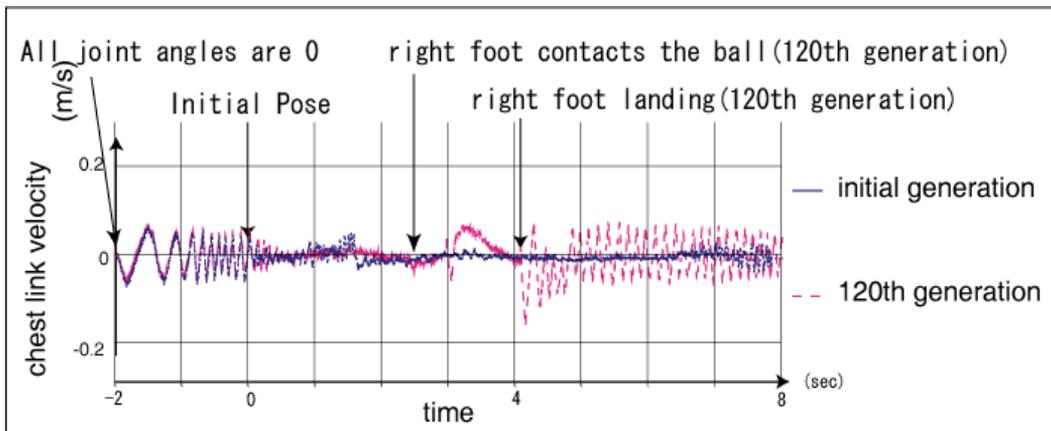


Figure 20. Variation of chest-link velocity

Figure 20 shows the chest-link velocity of x-axis. After the 120th generation, the robot learned to swing his leg so fast that the chest-link velocity was increased after the contact with the ball. Especially, the increase became large when the robot landed his foot on the floor. This is considered as a side effect of the fast swing. However, the robot never fell down because of the obtained motion that cancelled the moment by having the right hand swing down as the right foot swung forward, which is something that humans do (from 1.2 to 3.5 seconds in Fig. 19).

5. Discussion

5.1 Optimization using Multi-objective GA

In Section 4, GA was successfully applied to optimizing the humanoid motions. However, the task is essentially multi-objective optimization. For instance, in case of kicking, we have to consider the trade-off between the stability of the robot and the ball distance. Therefore, we have to apply Multi-objective GA (MOGA) (Fonseca & Fleming, 1993) with two fitness, i.e., the chest-link velocity (V_{\max}) and the ball travel distance (r). We empirically derive the velocity when the robot falls down and set the value as the velocity limitation of the chest link (V_{limit}). Rank-based fitness assignment method and niche-formation method are employed to determine the fitness for maximizing ($V_{\text{limit}} - V_{\max}$) and r . Roulette selection is also used according to the fitness values.

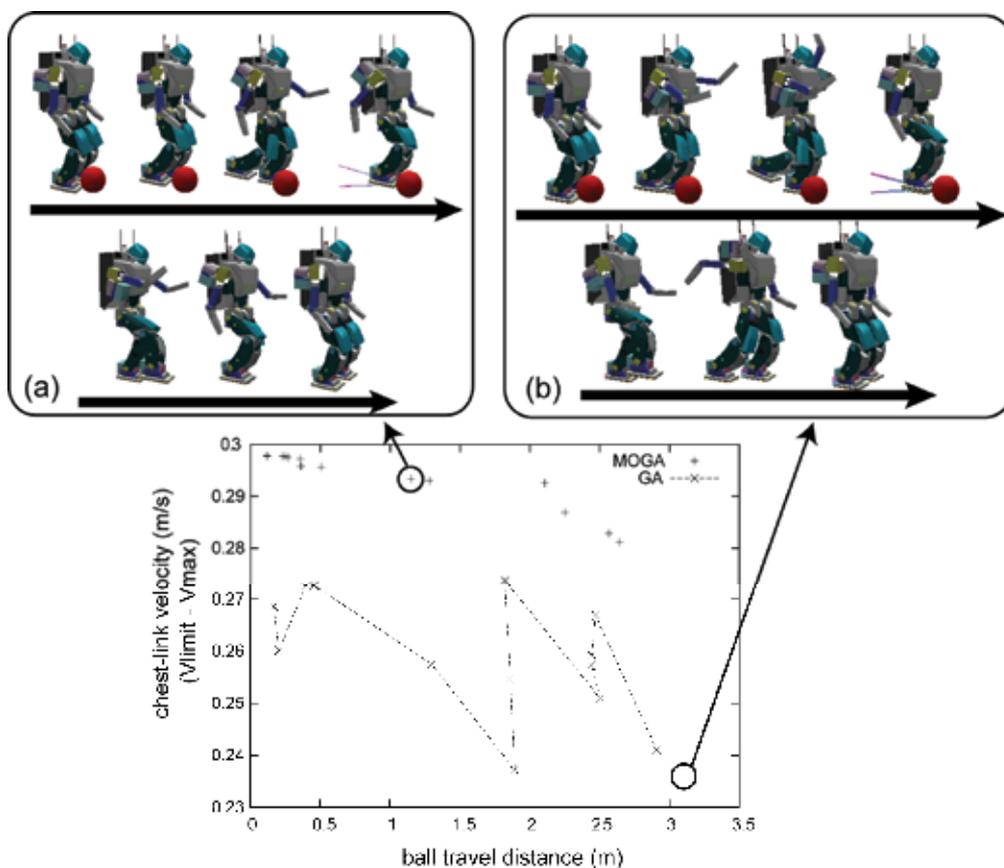


Figure 21. Fitness values of best individuals for Simple GA and Pareto-optimal individuals for MOGA

Fig. 21 shows the performance difference between simple GA and MOGA, in which 30 individuals were evolved for 50 generations. Best individuals at every generation are shown as the "x" dots for simple GA, whereas Pareto-optimal individuals are plotted as "+" dots

for MOGA. As can be seen from the figure, the chest-link velocity is smaller for MOGA when the robot tries to kick a ball at the same travel distance. This shows that MOGA, if applied effectively, is more suitable for designing humanoid motions. We will work on this topic, i.e., the extension of MOGA to a more complex task, for future research.

5.2 Future research

Currently, because the stability is not compensated using the controller of the actual robots, the motion range is restricted by the instability of the motion caused by the differences between the individuals. A future issue will be to compensate the stability using the actual robots, and thus to expand the range of motions that are feasible for the actual robots. In optimizing the kicks, the torque was not limited in the present experiment, so the results could not be applied to the actual robots without modification. In the future, our goal will be to apply these results to the actual robots, taking elements such as torque limits and angular velocity limits into consideration.

6. Conclusion

Through IEC, we proposed a motion design method for humanoid robots which does not need any specialized robotics knowledge, such as kinematics or dynamics. As an example, the designed dance motions were confirmed using the actual robots, and kicking motions were confirmed using a dynamics simulator. At the same time, however, in order to realize motions that deviate from the presuppositions of the physical model, the motions used with IEC were evolved using a GA.

In case of designing a sitting motion, it was possible to find the stable behaviour by searching for a space of joint angle trajectories. A kicking motion was set as a task, and the distance travelled by the ball was improved on a dynamics simulator. In addition, since the task is a multi-objective optimization, we empirically found MOGA successfully applicable to designing the motion.

Also, the ball travel distance and the stability, which we used as standards for evaluation, have a trade-off relationship with each other. In order to determine the optimum solution based on these two competing target functions, we would like to carry out optimization using a somewhat larger real-valued GA. Our future works include the application to more complicated tasks in consideration of physical constraints.

7. References

- Fonseca, C.M. & Fleming, P.J. (1993). Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp 416-423, Morgan Kaufmann
- Inoue, Y.; Tohge, T. & Iba, H. (2007). Cooperative transportation system for humanoid robots using simulation-based learning, *Applied Soft Computing*, Vol. 7, No.1, pp 115-125
- Inoue, Y.; Tohge, T. & Iba, H. (2004). Learning to acquire autonomous behavior - cooperation by humanoid robots, *Proceedings of the Genetic and Evolutionary Computation Conference*, pp 1394-1399
- Kanehiro, F.; Hirukawa, H. & Kajita, S. (2004). Open architecture humanoid robotics platform, *Journal of Robotics Research*, Vol.23, No.2, pp 155-165, October

- Kuroki, Y.; Blank, B.; Mikami, T.; Mayeux, P.; Miyamoto, A.; Playter, R.; Nagasaka, K.; Raibert, M.; Nagano, M. & Yamaguchi, J. (2003). Motion creating system for a small biped entertainment robot. *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, pp 1394-1399
- Nakaoka, S.; Nakazawa, A.; Yokoi, K. & Ikeuchi, K. (2003). Leg motion primitives for a humanoid robot to imitate human dances. *Proceedings of Sixth International Conference on Humans and Computers(HS2003)*, August
- Nishiwaki, K.; Kagami, S.; Kuniyoshi, Y.; Inaba, M. & Inoue, H. (2002). Online generation of humanoid walking motion based on a fast generation method of motion pattern that follows desired ZMP, *Proceedings of the 2002 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, EPFL, Lausanne, Switzerland, pp 2684-2689
- Takagi, H. (1998), Interactive evolutionary computation - cooperation of computational intelligence and human kansei, *Proceedings of 5th International Conference on Soft Computing and Information/Intelligent Systems*, pp 41-50, October
- Vukobratovic, M. & Stepanenko, J. (1972). On the stability of anthropomorphic systems, *Mathematical Biosciences*, Vol. 15, pp 1-37
- Wakaki, H. & Iba, H. (2002). Motion design of a 3D-CG avatar using interactive evolutionary computation, *Proceedings of 2002 IEEE international Conference on Systems, man and Cybernetics (SMC'02)*. IEEE Press
- Wolff, K. & Nordin, P. (2003). Learning biped locomotion from first principles on a simulated humanoid robot using linear genetic programming. *Proceedings of the Genetic and Evolutionary Computation Conference*, pp 495-506
- Yamane, K. & Nakamura, Y. (2002). Synergetic cg choreography through constraining and deconstraining at will, *Proceedings of International Conference on Robotics and Automation*, pp 855-862, May



Edited by Hitoshi Iba

This book presented techniques and experimental results which have been pursued for the purpose of evolutionary robotics. Evolutionary robotics is a new method for the automatic creation of autonomous robots. When executing tasks by autonomous robots, we can make the robot learn what to do so as to complete the task from interactions with its environment, but not manually pre-program for all situations. Many researchers have been studying the techniques for evolutionary robotics by using Evolutionary Computation (EC), such as Genetic Algorithms (GA) or Genetic Programming (GP). Their goal is to clarify the applicability of the evolutionary approach to the real-robot learning, especially, in view of the adaptive robot behavior as well as the robustness to noisy and dynamic environments. For this purpose, authors in this book explain a variety of real robots in different fields.

Photo by Ociacia / iStock

IntechOpen

