IntechOpen

Advanced Path Planning for Mobile Entities

Edited by Rastislav Róka

ADVANCED PATH PLANNING FOR MOBILE ENTITIES

Edited by **Rastislav Róka**

Advanced Path Planning for Mobile Entities
http://dx.doi.org/10.5772/intechopen.69591
Edited by Rastislav Róka

Contributors

Innocent Okoloko, Than Le, An T. Le, Nancy Arana-Daniel, Roberto Valencia-Murillo, Alma Y. Alanis, Carlos Lopez-Franco, Carlos Villaseñor, Lucía Hilario Pérez, Marta Covadonga Mora, Nicolás Montés Sánchez, Antonio Falcó Montesinos, Dora Luz Almanza Ojeda, Perla Lizeth Garza-Barrón, Mario-Alberto Ibarra-Manzano, Carlos Rubin Montoro-Sanjose, Xiangrong Xu, Gene Eu (Ching Yuh) Eu Jan, Chaomin Luo, Kai-Chieh Yang, Chi-Chia Sun

Notice

First published in London, United Kingdom, 2018 by IntechOpen
eBook (PDF) Published by IntechOpen, 2019
IntechOpen is the global imprint of INTECHOPEN LIMITED, registered in England and Wales, registration number: 11086078, The Shard, 25th floor, 32 London Bridge Street
London, SE19SG – United Kingdom
Printed in Croatia

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library

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

Advanced Path Planning for Mobile Entities
Edited by Rastislav Róka

p. cm.

Print ISBN 978-1-78923-578-4

Online ISBN 978-1-78923-579-1

eBook (PDF) ISBN 978-1-83881-400-7

Meet the editor

Rastislav Róka was born in Šaľa, Slovakia, on January 27, 1972. He received his MSc and PhD degrees in Telecommunications from the Slovak University of Technology, Bratislava, in 1995 and 2002. Since 1997, he has been working as a senior lecturer at the Institute of Multimedia Information and Communication Technologies, Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava. Since 2009, he has been working as an associated professor at this institute. His teaching and educational activities are realized in areas of fixed transmission media, designing, and planning of telecommunication networks and optical communication transmission systems. At present, his research activity is focused on signal transmission through optical transport and metropolitan and access networks using advanced optical signal processing including various multiplexing, modulation, and encoding techniques. His main effort is dedicated to effective utilization of the optical fiber's transmission capacity in broadband optical networks by means of dynamic bandwidth and wavelength allocation algorithms applied in various advanced hybrid passive optical network infrastructures.

Contents

Preface

The book *Advanced Path Planning for Mobile Entities* provides a platform for practicing researchers, academics, PhD students and other scientists to design, analyze, evaluate, process and implement diversiform issues of path planning, including algorithms for multipath and mobile planning and path planning for mobile robots. The nine chapters of the book demonstrate capabilities of advanced path planning for mobile entities to solve scientific and engineering problems with varied degree of complexity.

The first five chapters related to advanced algorithms for multipath planning provide details of methods for the consensus-based multipath planning with the collision avoidance applied in various environments and developed algorithms for search-based motion planning and path planning on quadric surfaces.

The second four chapters associated with extended path planning for mobile robots demonstrate possibilities of new approaches in path planning using neural network memory or parametric curves, motion planning, and navigation focused on mobile robots.

I hope that beginners and professionals in the field would benefit by going through the details given in the chapters of this book.

Rastislav Róka
Slovak University of Technology
Institute of MICT
FEI STU Bratislava, Slovakia

Section 1

Advanced Algorithms for Multi-Path Planning

Chapter 1

Consensus-Based Multipath Planning with Collision Avoidance Using Linear Matrix Inequalities

Innocent Okoloko

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.71288

Abstract

Consensus theory has been widely applied to collective motion planning related to coordinated motion. However, when the collective motion is highly irregular and adversarial, the basic consensus theory does not guarantee collision avoidance by default. As collision avoidance is a central problem of path planning, the incorporation of avoidance into the consensus algorithm is a subject of research. This work presents a new method of incorporating collision avoidance into the consensus algorithm, by applying the concept of constrained orientation control, where orientation constraints are represented as a set of linear matrix inequalities (LMI) and solved by semidefinite programming (SDP). The developed algorithm is used to simulate consensus-based multipath planning with collision avoidance for a team of communicating soccer robots.

Keywords: consensus, path planning, avoidance, optimization, LMI

1. Introduction

Path planning has found practical applications in areas such as entertainment (e.g. robot soccer) [1]; self-driving vehicles (e.g. Google's self-driving cars) [2]; intelligent highways [3], and multiple unmanned space systems [4]. Because of the potential applications, the topic of multipath planning has been studied extensively, for example in [5–11].

The simplicity and potential of consensus algorithms to generate *collective behaviors*, such as *flocking*, *platooning*, *rendezvous*, and other *formation* configurations, make it an attractive choice for solving certain problems in multiagent control. However, the basic consensus algorithm collision avoidance mechanism is not developed for *adversarial* situations (i.e., opposite or attacking motion). To extend the power of the algorithm, it is therefore necessary to develop more powerful collision avoidance capabilities.

Next, we consider the basic approaches to collision avoidance in consensus. Some researchers, for example, [12, 13], approached the avoidance problem by introducing potential forces such as attraction and repulsion. However, the potential force algorithms were not developed for adversarial reconfigurations, for example, vehicles moving in opposite directions. Potential functions also have a problem of getting into local minima, coupled with slow speed of convergence. It is observed in [12] that any repulsion based on potential functions alone is not sufficient to guarantee consensus-based collision avoidance. Moreover, the attitude change maneuver presented in [12] was not developed for three-dimensional space (see [14] for a comprehensive literature survey on this topic).

Thus, in this work, we present an approach which we previously developed [5, 9] for incorporating collision avoidance into the consensus framework by applying quadratically constrained attitude control (Q-CAC), via semidefinite programming (SDP), using linear matrix inequalities (LMI). The main benefit of this approach is that it can solve the collision avoidance problem in adversarial situations and any configurations, and the formulation can be applied to two-dimensional as well as three-dimensional spaces. **Table 1** shows the notation frequently used in this chapter.

Notation	Meaning
x^i	Position vector of vehicle number i
$(x^{ij})^{off}$	Offset vector of vehicles i and j
$\mathbf{x}$	Stacked vector of more than one position vector
$\mathbf{x}^{off}$	Stacked vector of more than one offset vector
$u^i, \dot{x}^i$	Control input of vehicle i
$\mathbf{u}, \dot{\mathbf{x}}$	Stacked vector of control inputs of more than one vehicle
$\mathbf{L}$	Laplacian matrix
$\boldsymbol{S}^m$	The set of $m \times m$ positive-definite matrices
$\mathbf{S}$	Bounding sphere or circle of a vehicle or obstacle
ε	Width of safety region
r^*	Radius of $\mathbf{S}$
r	$r^* + \varepsilon$
v^i	Attitude vector of vehicle i
v^i_{obs}	Obstacle vector of vehicle i
v^{ij}_{obs}	Obstacle vector of vehicle i emanating from vehicle j
D^{ij}	Euclidean distance between vehicles i and j
L^{ij}	Line passing through the mid points of vehicles i and j
ρ^{ij}	Perpendicular bisector of L^{ij} separating vehicles i and j
PL^i	Plane passing through the midpoint of vehicle i
l^{ij}	Line of intersection of PL^i and PL^j
d^i_x	Distance from x^i to l^{ij} (for 3D) or p^{ij}(for 2D)

Notation	Meaning
d_v^i	Distance from v^i to l^{ij} (for 3D) or p^{ij}(for 2D)
z^i	A point on the Z axis of PL^i
p^{ij}	Point of intersection of the lines passing through $\overrightarrow{x^i(t)v^i(t)}$ and $\overrightarrow{x^j(t)v^j(t)}$
N^i	Normal vector perpendicular to x^i, v^i, and z^i
$\mathbf{D}$	Attitude control plant matrix, $\mathbf{D} \in \mathcal{S}^m$
$\otimes$	Kronecker multiplication operator
$\mathbf{A}$	State or plant matrix for dynamics of $\mathbf{x}$
$\mathbf{B}$	Input matrix for dynamics of $\mathbf{x}$ for input $\mathbf{u}$
$\mathbf{F}$	Feedback controller matrix
$\mathbf{K}$	Proportional constant
$\mathbf{I}_p$	Identity matrix of size $p \times p$
Γ	$\Gamma = \mathbf{L} \otimes \mathbf{I}_p$
$\mathbf{H}$	A vector or matrix in the Schur inequality
$\mathbf{R}$	A positive-definite matrix in the Schur inequality
$\mathbf{Q}$	A symmetric matrix in the Schur inequality
η	Positive real number for scaling the consensus term
β	Positive real number for scaling the proportional term

Table 1. Frequently used notation in this chapter.

2. Problem statement

The basic *consensus* problem is that of driving the states of a team of communicating agents to a common value by distributed protocols based on their *communication graph*. The agents (or vehicles) $i (i = 1, \cdots, n)$ are represented by vertices of the graph, whereas the edges of the graph represent communication links between them. Let x_i denote the state of a vehicle i and $\mathbf{x}$ is the stacked vector of the states of all vehicles. For systems modeled by first-order dynamics, the following first-order consensus protocol (or its variants) has been proposed, for example in [12, 13]

$$\dot{\mathbf{x}}(t) = -\mathbf{L}\left(\boldsymbol{x}(\boldsymbol{t}) - \boldsymbol{x}^{\boldsymbol{off}}\right). \tag{1}$$

Consensus is said to have been achieved when $\|x^i - x^j\| \to (x^{ij})^{off}$, as $t \to \infty$, $\forall i \neq j$.

The *consensus-based multipath planning with collision avoidance* problem can be stated as follows: Given a set of vehicles i, with initial positions $x^i(t_0)$, desired final positions x_d^i, at time t_f, a set of obstacles with positions $x_{obs}^j (j = 1, \cdots, m)$, and the Laplacian matrix of their communication graph $\mathbf{L}$ find a sequence of collision-free trajectories from t_0 to t_f such that $x^i\left(t_f\right) = x_d^i \forall i$. Protocol (Eq. (1)) on its own does not solve the collision avoidance problem in adversarial

situations. A comprehensive presentation of the necessary mathematical tools for this work (including graph theory and consensus theory) can be found in [14].

3. Solutions

In this section, we develop solutions to the problem stated in Section 2.

3.1. Consensus-based arbitrary reconfigurations

It was shown that for the dynamic system.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \tag{2}$$

there exists a stabilizing feedback controller **F**, such that the protocol

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{F}\mathbf{u} \tag{3}$$

drives $\mathbf{x}$ to $\mathbf{x}_f$ [15]. Here, $\mathbf{x} = [x^1, \cdots, x^n]$ is a stacked vector of the initial positions of the vehicles, $\mathbf{u} = -\Gamma(\mathbf{x} - \mathbf{x}^{off})$, $\Gamma = \mathbf{L} \otimes \mathbf{I}_p$, $\mathbf{I}_p$ is the identity matrix of size $p \times p$, and p is the state dimension of the vehicles.

To begin, we first consider the *reference consensus path planning* problem. To this end, the following protocol is proposed for a *leader-follower* communication graph architecture

$$\mathbf{u} = -\Gamma\left(\mathbf{x} - \mathbf{x}^{off}\right) + \mathbf{K}\left(\mathbf{x}^{off} - \mathbf{x}\right). \tag{4}$$

The corresponding protocol for a *leaderless* architecture is

$$\mathbf{u} = -\Gamma\left(\mathbf{x} - \mathbf{x}^{off}\right) + \mathbf{K}(\mathbf{x}_d - \mathbf{x}), \tag{5}$$

where $\boldsymbol{x}_d \neq \boldsymbol{x}^{off}$ is the desired final position and is different from the formation configuration, $\mathbf{K} = \boldsymbol{\epsilon}\mathbf{I}_n$, $(0 < \boldsymbol{\epsilon} \ll 1)$, and n is the dimension of $\mathbf{x}$.

Theorem 1 The time-varying system (Eq. (2)) achieves consensus.

Proof: see [14].

Figure 1 shows a simulation of consensus-based reconfiguration, using the communication graph in **Figure 2**, which is an example of a *leader-follower* graph. Node 1 is the *leader,* and each of the other nodes is connected to their adjacent neighbors. In **Figure 1**, the dots inside small circles indicate initial positions, whereas the dot in the diamond is the initial position of the leader. The stars indicate desired final positions. The larger circles with dashed lines are positions where collisions occurred, and the diameters of the circles indicate the size of intersection of the safety regions of the vehicles. The simulation proves that for arbitrary reconfigurations, the basic consensus algorithm does not guarantee collision avoidance.

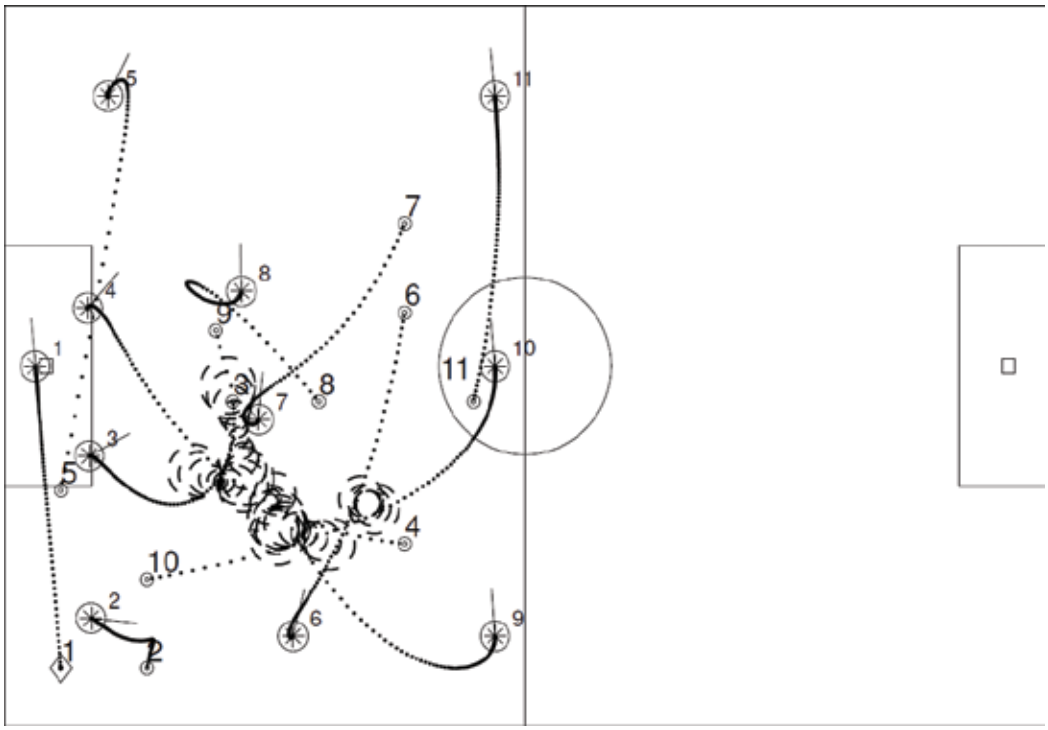

Figure 1. Consensus-based reconfiguration in adversarial situation using topology.

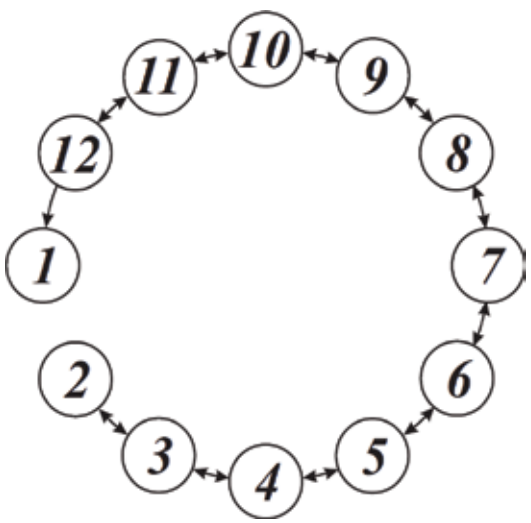

Figure 2. Topology: a leader-follower graph.

3.2. Quadratically constrained attitude control-based collision avoidance

The collision avoidance problem is that of avoiding static obstacles and other moving vehicles while driving the state of a vehicle from one point to another. For simplicity, we approximate a vehicle or an obstacle by **S**, as shown in **Figure 3**. A nonspherical obstacle may be represented by a polygon as shown in **Figure 4**. For the **S**-type obstacle (or vehicle), let the obstacle be centered on a point x_{obs}; it is desired that the time evolution of any vehicle state $x^i(t)$ from t_0 to t_f should avoid the constraint region shown in **Figure 3**.

The feasible region is thus defined by

$$x_{feas} = \{x \in \mathbb{R}^{m\times m} | \|x - x_{obs}\| > r^*\}, \quad m \in \mathbb{R}, \tag{6}$$

where r^* is the radius of **S**, bounded by a safety region of width ε.

There is no direct representation of the nonlinear nonconvex equation (Eq. (6)) as LMI. However, some non-LMI methods, for example, mixed integer linear programming (MILP) [7],

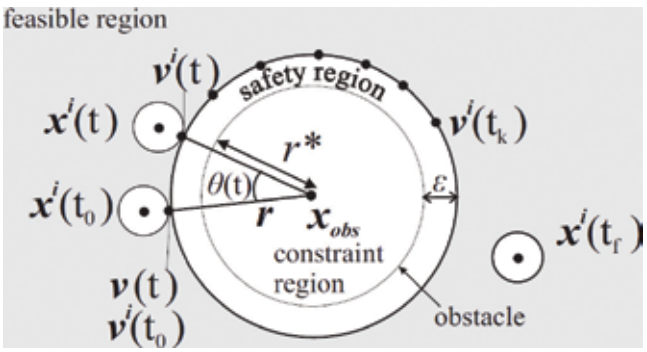

Figure 3. Constrained control problem for a static spherical obstacle.

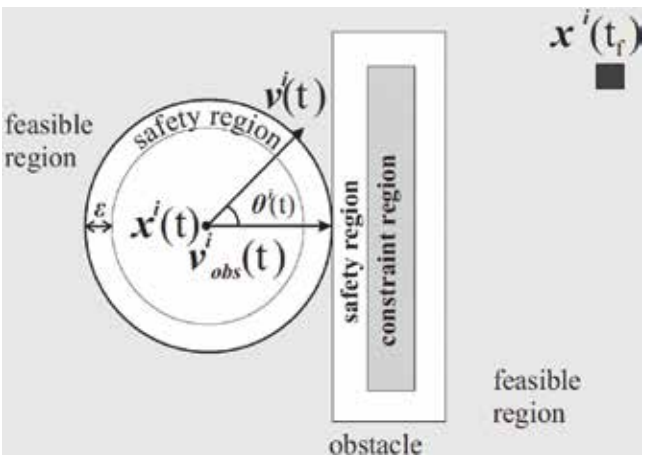

Figure 4. Constrained control problem for static nonspherical obstacle.

have been developed for approximating its solution. In this section, we present an approach, which we previously developed in [5, 9, 10, 14], based on the principles of quadratically constrained attitude control (Q-CAC) algorithm [16], initially developed for the spacecraft attitude control problem.

At any time t, suppose the safety region of vehicle i centered on $x^i(t)$ intersects the safety region of an obstacle, *obs*, centered on x_{obs}. Let $v(t)$ be the unit vector extending from the centre of x_{obs} or $x^i(t)$ in the direction of the point of intersection. The vectors $v(t)$ will be different for each vehicle or obstacle. Considering the case shown in **Figure 3**, assume x_{obs} is known and $v(t)$ is also known in the frame of *obs*. Then, to guide vehicle i safely around the obstacle, define a unit vector $v^i(t)$ in the direction of $v(t)$ in the frame of *obs*. The vector $v^i(t)$ will be regarded as an imaginary vector whose direction can be constrained to change with time. The vector $v^i(t)$ can then be used to find a sequence of trajectories around *obs* which guides i from $x^i(t_0)$ to $x^i(t_f)$ without violating (Eq. (6)).

The problem reduces to the Q-CAC problem. It is desired that the angle θ between $v^i(t)$ and $v(t)$ should be larger than some given angle $\varnothing$, $\forall t$. The constraint is

$$v^i(t)^T v(t) \leq \cos\varnothing, \forall t \in [t_0, t_f]. \tag{7}$$

The idea is to control the angle between the unit vectors $v^i(t)$ and $v(t)$. This implies that one of the vectors $v^i(t)$ or $v(t)$ must remain static, whereas the other moves with time. Vector $v^i(t)$ is used to control the position of the vehicle; therefore, $v^i(t)$ will move with time. The positions of $v^i(t)$ define a trajectory path for $x^i(t)$. Thus, $x^i(t)$ is forced to move on the surface of the safety region bounding **S**. At some time t_k, $x^i(t)$ will arrive close to a point indicated by $v^i(t_k)$, at which a translation to $x^i(t_f)$ is unconstrained. This is shown by the black dots on the boundary of the safety region in **Figure 4**. To obtain the unit vector $v(t)$, the actual vector extending from the centre of x_{obs} or $x^i(t)$ in the direction of the point of intersection is normalized. After the solution $v^i(t)$ is obtained as a unit vector, $v^i(t)$ is multiplied by $r = r^* + \varepsilon$ to obtain the actual safe trajectory.

Let $\mathbf{v}(t) = [v^i(t)^T v(t)^T]^T$, then the dynamics of $\mathbf{v}(t)$ is defined as

$$\dot{\mathbf{v}}(t) = \mathbf{D}(t)\mathbf{v}(t), \tag{8}$$

where $\mathbf{D} \in \mathcal{S}^{pn}$, p is the dimension of the state vector x^i, and n is the number of vehicles. The above differential equation represents the rotational dynamics of the two vectors contained in $\mathbf{v}(t)$. **D** is a semidefinite matrix variable whose contents are unknown. Its purpose is to vary the angle between the two vectors in $\mathbf{v}(t)$ with time while also keeping them normalized.

The discrete time equivalent of the above differential equation is

$$\mathbf{v}(k+1) = \Delta t \mathbf{D}(k)\mathbf{v}(k), \tag{9}$$

where $k = 0, \cdots, N$ ($N\Delta t = t_f$) is the discrete time equivalent of t and Δt is the discretization time-step. To implement Eq. (9), **D** is declared in a semidefinite program which chooses the appropriate values to rotate the vectors in $\mathbf{v}(t)$ while satisfying norm constraints. Note in the above discretization of the differential equation, the identity matrix cannot be added to the solution;

instead, the matrix **D** is chosen implicitly to satisfy the rotation. The vectors in **v**(*t*) are unit vectors; they are not translating, but they are rotating and must be preserved as unit vectors.

To enforce the attitude constraint (Eq. (7)) in a SDP, it should be represented as a LMI using the *Schur complement formula* described in [17]. The Schur complement formula states that the inequality

$$\mathbf{H}\mathbf{R}^{-1}\mathbf{H}^{\mathbf{T}} - \mathbf{Q} \leq 0, \tag{10}$$

where $\mathbf{Q} = \mathbf{Q}^{\mathbf{T}}$, $\mathbf{R} = \mathbf{R}^{\mathbf{T}}$, and $\mathbf{R} > 0$ are equivalent to and can be represented by the linear matrix inequality

$$\begin{bmatrix} Q & H \\ H^T & R \end{bmatrix} \geq 0. \tag{11}$$

Note that Eq. (7) is equivalent to

$$\underbrace{\begin{bmatrix} v^i(t)^T & v(t)^T \end{bmatrix}}_{v(t)^T} \begin{bmatrix} \mathbf{0}_3 & \frac{1}{2}\mathbf{I}_3 \\ \frac{1}{2}\mathbf{I}_3 & \mathbf{0}_3 \end{bmatrix} \begin{bmatrix} v^i(t) \\ v(t) \end{bmatrix} \leq \cos \varnothing, \tag{12}$$

which also implies that

$$\underbrace{\begin{bmatrix} v^i(t)^T v(t)^T \end{bmatrix}}_{v(t)^T} \underbrace{\begin{bmatrix} \mathbf{0}_3 & \mathbf{I}_3 \\ \mathbf{I}_3 & \mathbf{0}_3 \end{bmatrix}}_{\mathbf{G}} \begin{bmatrix} v^i(t) \\ v(t) \end{bmatrix} \leq 2\cos \varnothing, \tag{13}$$

Note also that some of the eigenvalues of the **G** in Eq. (13) are nonpositive. To make the matrix positive definite, one only needs to shift the eigenvalues of **G**, by choosing a positive real number μ which is larger than the largest absolute value of the eigenvalues of **G**, then

$$\underbrace{\begin{bmatrix} v^i(t)^T & v(t)^T \end{bmatrix}}_{v(t)^T} \left(\mu \mathbf{I}_6 + \begin{bmatrix} \mathbf{0}_3 & \mathbf{I}_3 \\ \mathbf{I}_3 & \mathbf{0}_3 \end{bmatrix} \right) \underbrace{\begin{bmatrix} v^i(t) \\ v(t) \end{bmatrix}}_{v(t)} \leq 2\left(\cos \varnothing + \mu \right). \tag{14}$$

Let $\mathbf{M} = \left(\mu \mathbf{I}_6 + \begin{bmatrix} \mathbf{0}_3 & \mathbf{I}_3 \\ \mathbf{I}_3 & \mathbf{0}_3 \end{bmatrix} \right)^{-1}$, then **M** is positive definite. Therefore, following the Schur complement formula, the LMI equivalent of Eq. (14) is

$$\begin{bmatrix} 2\left(\cos \varnothing + \mu \right) & \boldsymbol{v}(\boldsymbol{t})^T \\ \mathbf{v}(\boldsymbol{t}) & \mathbf{M} \end{bmatrix} \geq 0. \tag{15}$$

For collision avoidance, the dynamic system (Eq. (8)) is solved whenever it is required, subject to the attitude constraint (Eq. (15)) and norm constraints $\|v^i(t)\| = 1$ and $\|v(t)\| = 1$. Thus, the

optimization problem of collision avoidance is essential to find a feasible v^i subject to the following constraints:

$$\mathbf{v}_{k+1} = \Delta t \mathbf{D}(t)\mathbf{v}_k, \tag{16}$$

$$\mathbf{v}_k^T(\boldsymbol{v}_{k+1} - \boldsymbol{v}_k) = 0, \tag{17}$$

$$\begin{bmatrix} 2(\cos\varnothing + \mu) & \mathbf{v}(\boldsymbol{t})^T \\ \mathbf{v}(\boldsymbol{t}) & \mathbf{M} \end{bmatrix} \geq 0. \tag{18}$$

Eq. (17) is essentially the discrete time version of $\mathbf{v}(t)^T\dot{\mathbf{v}}(t) = 0$ which guarantees that $\mathbf{v}(t)^T\mathbf{v}(t) = 2$, if $\|v^i(0)\| = 1$ and $\|v(0)\| = 1$. This solution works for 2D and 3D spaces. The next step is to extend the formulation to the case of dynamic obstacles. First, consider two vehicles i and j with states $x^i(t)$, $x^j(t)$ and attitude vectors $v^i(t)$, $v^j(t)$, respectively. Collision avoidance requires that they must avoid each other always. As shown in **Figure 5**, any time their safety regions are violated and the point of their intersection in the coordinate frame of i is $v^i_{obs}(t)$.

The avoidance requirements are

$$\theta^i(t) \geq \varnothing \equiv v^i(t)^T v^i_{obs}(t) \leq \cos\varnothing, \tag{19}$$

$$\theta^j(t) \geq \varnothing \equiv v^j(t)^T v^j_{obs}(t) \leq \cos\varnothing, \tag{20}$$

$$\forall t \in [t_0, t_f],$$

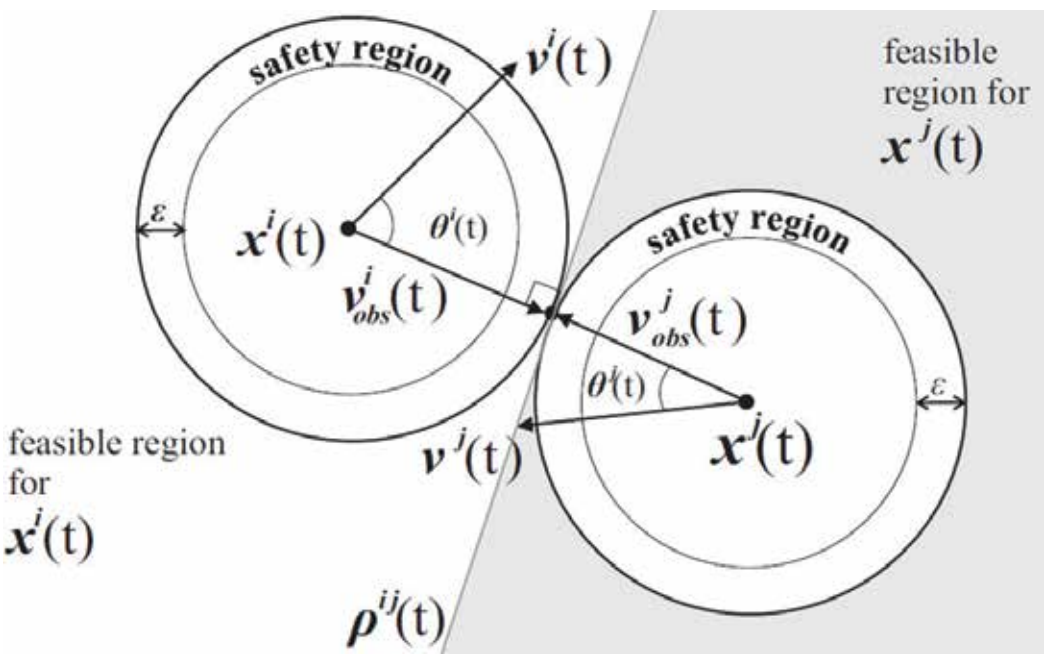

Figure 5. Constrained control problem for dynamic obstacles.

where $\varnothing \geq \pi/2$. For this dynamic situation, it is sufficient to enforce the following avoidance constraints:

$$\begin{bmatrix} 2(\cos\varnothing + \mu) & \begin{bmatrix} v^i(k+2) \\ v^i_{obs}(k+2) \end{bmatrix}^T \\ \begin{bmatrix} v^i(k+2) \\ v^i_{obs}(k+2) \end{bmatrix} & \mathbf{M} \end{bmatrix} \geq 0, \tag{21}$$

$$\begin{bmatrix} 2(\cos\varnothing + \mu) & \begin{bmatrix} v^j(k+2) \\ v^j_{obs}(k+2) \end{bmatrix}^T \\ \begin{bmatrix} v^j(k+2) \\ v^j_{obs}(k+2) \end{bmatrix} & \mathbf{M} \end{bmatrix} \geq 0, \tag{22}$$

$$i, j = 1, \cdots, n, i \neq j.$$

Note that $(k+2)$ is used because the optimization is performed two steps ahead of time to ensure that the future trajectories are collision free. However, when this avoidance protocol is applied to dynamic collision avoidance, some vehicle configurations pose challenges and this is considered next.

3.3. Conflict resolution for multiple vehicles

A collision between two vehicles i and j is imminent at time t whenever

$$D^{ij}(t) = \left\| x^i(t) - x^j(t) \right\| \leq \left(r^i + r^j \right), \tag{23}$$

which can be computed using position feedback data determined by onboard or external sensors or communicated among the vehicles.

There are two aspects of collision problems: (i) *collision detection* and (ii) *collision response*. Collision detection is the computational problem of detecting the intersection of two or more objects. This can be done either using sensors or numerically using concepts from linear algebra and computational geometry. Collision response is the initiation of the appropriate avoidance maneuver. In this section, we present methods to detect different configurations of collisions and classify them. Then, an appropriate response technique is developed for each of the collision configurations.

Consider two vehicles i and j, whose current states are $x^i(t)$ and $x^j(t)$ and the desired final states are $x^i(t_f)$ and $x^j(t_f)$. We identify three different basic collision configurations as: (i) *simple collision*; (ii) *head-on collision*; and (iii) *cross-path collision*. Solutions will be developed for each of these configurations, and when combined synergistically, they will provide sufficient collision avoidance behavior for fast collision-free reconfiguration for the team of vehicles.

3.3.1. Detecting and resolving a simple collision

A simple collision problem is any configuration in which $D^{ij}(t) \leq (r^i + r^j)$ and the current vector directions (or attitude vectors) $v^i(t)$ and $v^j(t)$ of vehicles i and j are on different sides of the plane or infinite line $L^{ij}(t)$ passing through the points $x^i(t)$, $x^j(t)$; and the attitude vectors $\overrightarrow{x^i(t)v^i(t)}$ and $\overrightarrow{x^j(t)v^j(t)}$ are not parallel. Note that when $\overrightarrow{x^i(t)v^i(t)}$ and $\overrightarrow{x^j(t)v^j(t)}$ are not parallel, a point or line of intersection can be computed for both vectors. Examples of simple collision problems are shown in **Figures 5** and **6**.

This is the easiest collision problem to solve because the attitude vectors are already on opposite sides of $L^{ij}(t)$. Considering **Figure 6** (b), the plane or line $\rho^{ij}(t)$ tangent to the point of intersection of both vehicles constrains the current motion spaces of the vehicles to either of the two sides of the plane at time t. A pure optimization-based solution will attempt to search the space on the right side of $\rho^{ij}(t)$ to seek for a point which is closest to the goal of i, and this will be used as the next trajectory. The algorithm will also search the left side of $\rho^{ij}(t)$ to find the next trajectory for j. Once the positions are updated, a new $\rho^{ij}(t)$ is computed.

Indeed, the solution is provided by the basic collision avoidance protocols (Eqs. (21) and (22)) without having to do a set search. It is easy to observe that by expanding the angles $\theta^i(t)$ and $\theta^j(t)$ and choosing the next feasible trajectories $r^*/2$ along the new direction vectors $\overrightarrow{x^i(t)v^i(t)}$ and $\overrightarrow{x^j(t)v^j(t)}$, the new trajectories are bound to satisfy the feasible regions separated by $\rho^{ij}(t)$, provided $\varepsilon^i > r^{*i}$ for any i. The rest of the avoidance strategies developed in the remaining part of this section are attempts to reduce more complex collision configurations to a simple collision configuration.

3.3.2. Detecting and resolving a head-on collision

A head-on collision problem is any configuration in which $D^{ij}(t) \leq (r^i + r^j)$ and $v^i(t)^T v^j(t) \approx \pi$ rad. **Figure 7(a)** illustrates the head-on collision problem.

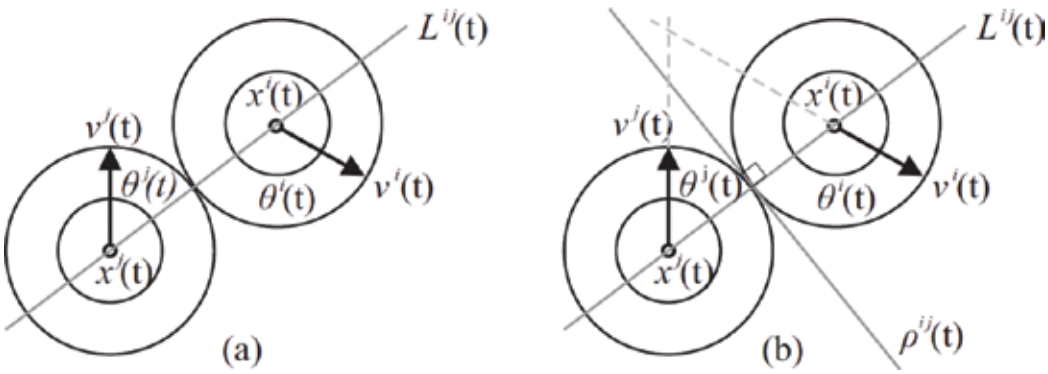

Figure 6. Simple collision problem.

The paths from the current positions $x^i(t)$ and $x^j(t)$ to the goal positions $x^i(t_f)$ and $x^j(t_f)$ lead to a configuration in which the attitude vectors $\overrightarrow{x^i(t)v^i(t)}$ and $\overrightarrow{x^j(t)v^j(t)}$ are parallel (or close to parallel) and in opposite directions, in the sense that a point of intersection cannot be computed. **Figure 7(b)–(d)** shows several examples of head-on collision. **Figure 7(b)** is a *direct head-on collision* because the vectors $\overrightarrow{x^i(t)v^i(t)}$ and $\overrightarrow{x^j(t)v^j(t)}$ are lying directly on $L^{ij}(t)$. **Figure 7(c)** is an *approximate head-on collision* and **Figure 7(d)** is a head-on collision that can be easily converted to a simple collision configuration.

For the configurations in **Figure 7(b)** and **(c)**, the Q-CAC formulation presented earlier easily solves this problem without any modifications to the algorithm. However, whenever $v^i(t)^T v^i_{obs}(t) \approx 0$ for any i, the optimization algorithm takes some significant time to solve. Even though the resulting trajectory is desirable, this delay is undesirable for real-time collision avoidance. Therefore, whenever this configuration is encountered for any two vehicles, a one-step elementary evasive maneuver is initiated, in which either $v^i(t)$ or $v^j(t)$ is rotated by a small angle $\psi > 0$. This rotation effectively transforms the head-on collision

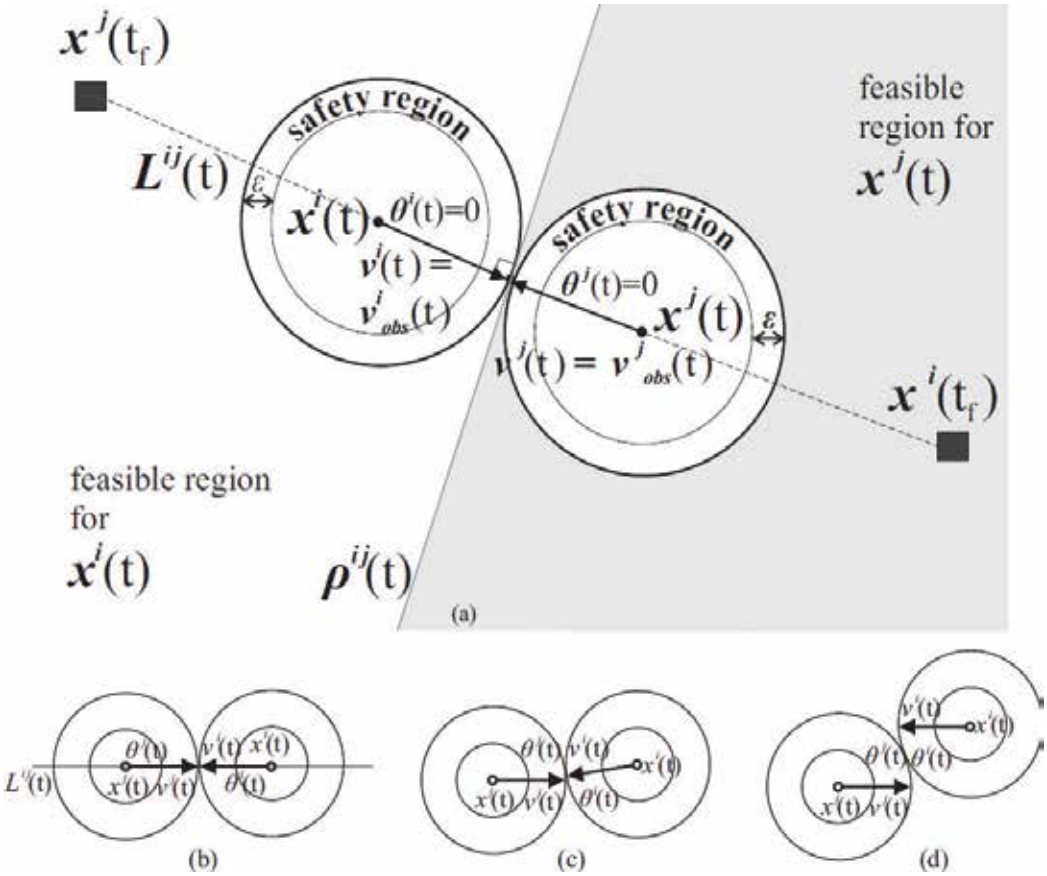

Figure 7. Head-on collision problem.

configuration to a simple collision configuration. Once this is done, the avoidance constraints defined in Eqs. (21) and (22) solve in real time. The trajectory obtained using this strategy for two-vehicle reconfiguration with head-on collision avoidance is shown in [14].

3.3.3. Detecting and resolving cross-path collision for two vehicles

A cross-path collision problem is any configuration in which $D^{ij}(t) \leq (r^i + r^j)$ and the current vector directions $v^i(t)$ and $v^j(t)$ are on the same side of $L^{ij}(t)$, and the attitude vectors $\overrightarrow{x^i(t)v^i(t)}$ and $\overrightarrow{x^j(t)v^j(t)}$ are not parallel. Because the vectors are not parallel, a point (for 2D) or line (for 3D) of intersection can be computed for both vectors. **Figure 8** is an example of a cross-path collision problem.

Note that for the avoidance process, the attitude control algorithm attempts to expand the angles $\theta^i(t)$ and $\theta^j(t)$ to an angle = $\pi/2$. Based on this initial configuration, therefore, $v^i(t)$ and $v^j(t)$ will remain parallel or close to parallel, but not in opposite directions. If this continues, the desired goal positions may never be reached, or may be reached after a great deal of effort. To resolve this problem, it is required to determine whether the two vehicles are indeed in a cross-path configuration. The task is therefore to see if there exists a point or line of intersection between $\overrightarrow{x^i(t)v^i(t)}$ and $\overrightarrow{x^j(t)v^j(t)}$, and if such an intersection lies on one side of L^{ij}.

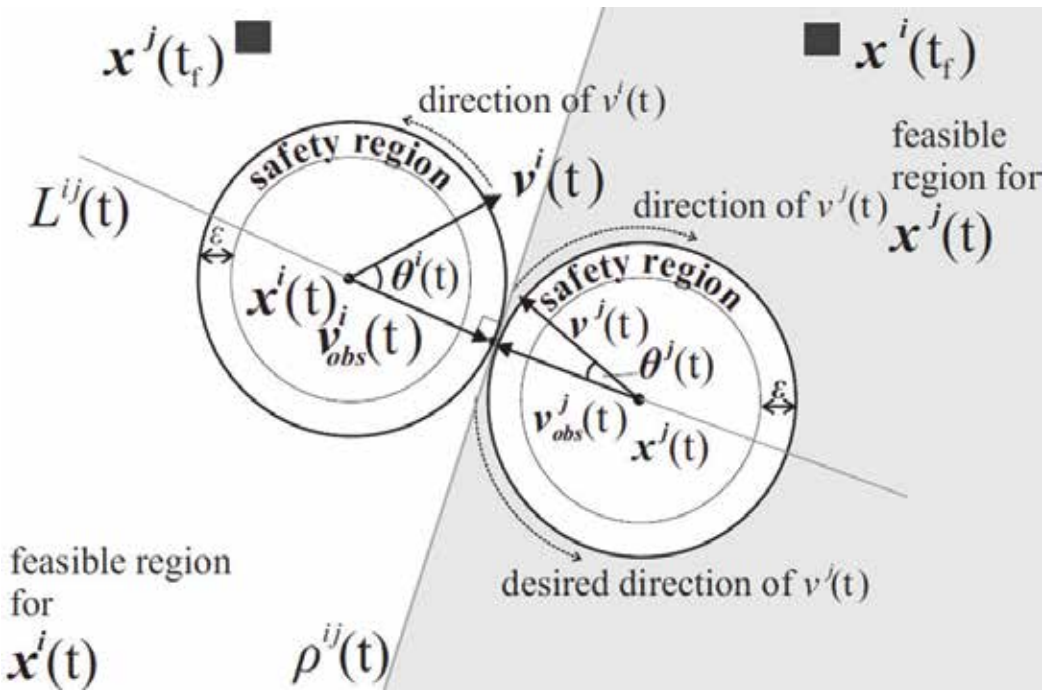

Figure 8. Cross-path collision trajectory.

3.3.4. Determining cross-path collision in 3D and 2D.

To determine cross-path collision between i and j in 3D, two planes PL^i and PL^j are defined, both parallel to the z axes of the world coordinate frame (**Figure 9**). Each plane must contain the vectors $\overrightarrow{x^i(t)v^i(t)}$ and $\overrightarrow{x^j(t)v^j(t)}$ as shown in the figure. Therefore, the plane PL^i is defined as the set $(N^i(t), x^i(t), v^i(t), z(t))$, where $z^i(t)$ is a point chosen above or below $x^i(t)$ or $v^i(t)$ on the z axis and $N^i(t)$ is the normal vector perpendicular to $x^i(t)$, $v^i(t)$, and $z^i(t)$. Once $N^j(t)$ is similarly defined, the intersection of the two planes can be computed using techniques from computational geometry. If the two planes are not parallel, the computation of planes' intersection will return a line l^{ij}. Once this line is determined, the next step is check if it is on one side of the plane parallel to the z axis and containing the points $x^i(t)$ and $x^j(t)$.

An easy way to do this is to compute the perpendicular distances from the points $x^i(t)$, $v^i(t)$, $x^j(t)$, and $v^j(t)$, to l^{ij}.

Let the corresponding distances be:

$$d_x^i(t) = \left\| x^i(t) - l^{ij} \right\|, \tag{24}$$

$$d_v^i(t) = \left\| v^i(t) - l^{ij} \right\|, \tag{25}$$

$$d_x^j(t) = \left\| x^j(t) - l^{ij} \right\|, \tag{26}$$

$$d_v^j(t) = \left\| v^j(t) - l^{ij} \right\|. \tag{27}$$

If $d_v^i(t) \le d_x^i(t)$ and $d_v^j(t) \le d_x^j(t)$, then the line of intersection is in front of both vehicles, and a cross-path collision is imminent as shown in **Figure 9(a)** and **(b)**. Otherwise, there is no cross-path conflict as shown in **Figure 9(c)**.

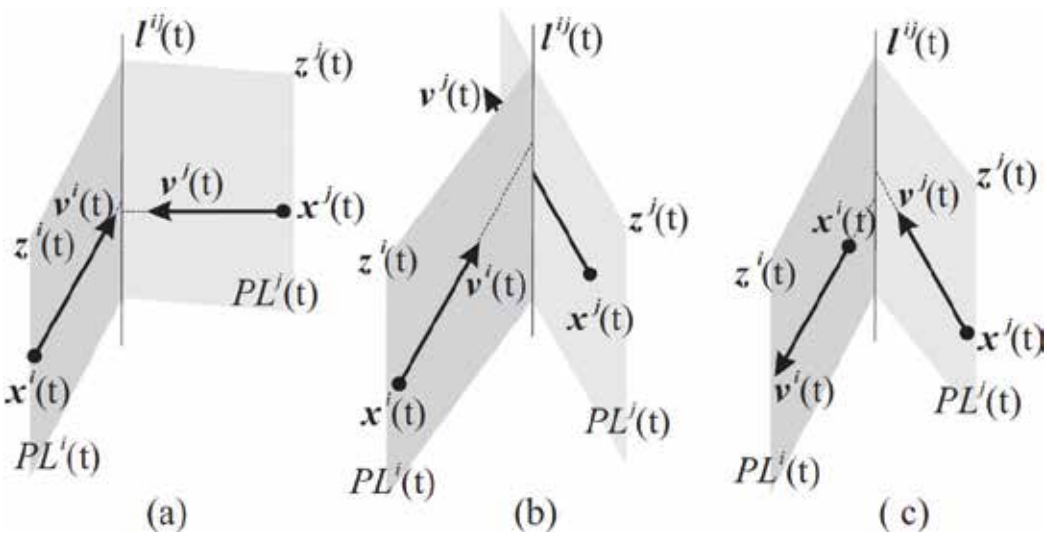

Figure 9. Determination of cross-path collision in 3D.

The analysis is simpler in the 2D case. Instead of l^{ij}, we search for a point p^{ij}, which is the point of intersection of the lines passing through $\overrightarrow{x^i(t)v^i(t)}$ and $\overrightarrow{x^j(t)v^j(t)}$ as shown in **Figure 10**. If indeed such a point is found, we use p^{ij}instead of l^{ij}in the previous set of equations.

Figure 11 shows an illustration of the computation of d_x^i and d_v^i for any i.

Figure 12(a) is a cross-path collision configuration, but (b) is a simple collision configuration.

The solution strategy adopted is to convert any cross-path configuration such as **Figure 12** (a) to a simple configuration such as (b). To do this, one only must move either $v^i(t)$ or $v^j(t)$ to the other side of $L^{ij}(t)$ (or onto the line $L^{ij}(t)$). A simple strategy to decide which $v(t)$ should be

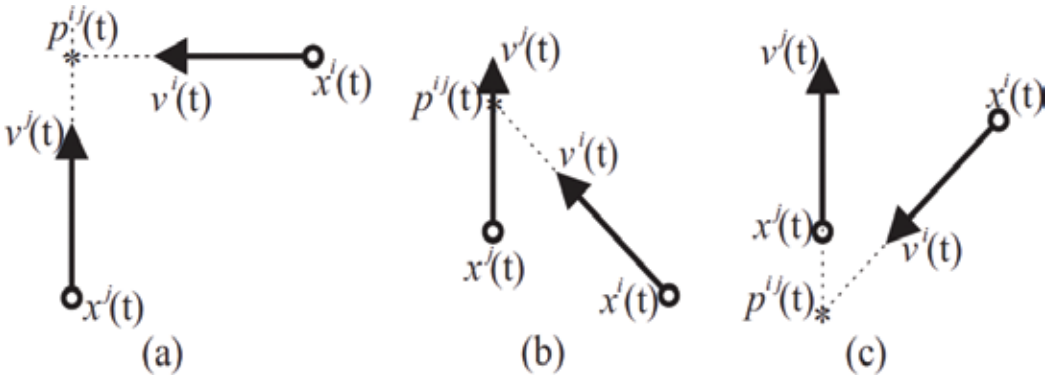

Figure 10. The point p^{ij} is the point of intersection of the infinite lines passing through direction vectors $\overrightarrow{x^i(t)v^i(t)}$ and $\overrightarrow{x^j(t)v^j(t)}$. The position of p^{ij} in relation to both direction vectors determines if a cross-path collision is imminent. If p^{ij} is in front of both vectors as in (a) and (b), then a cross-path collision is imminent; otherwise, no cross-path collision is imminent as in (c).

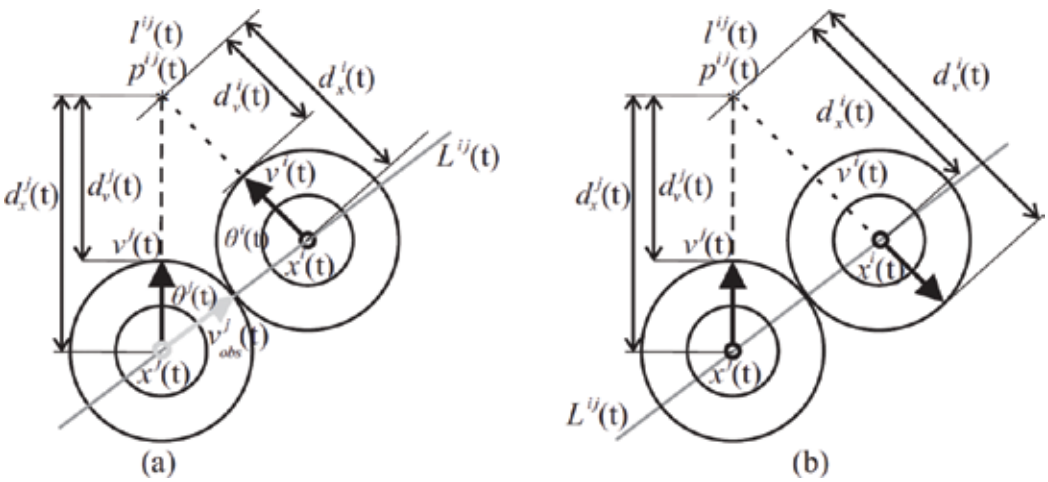

Figure 11. Continuing the explanation from **Figure 10**, d_x^i is the distance from any x^i (vehicle i) top^{ij}, whereas d_v^i is the distance from v^i to p^{ij}, that is, the distance of the outer boundary (where v^i lies) of the safety region of vehicle i to p^{ij}.

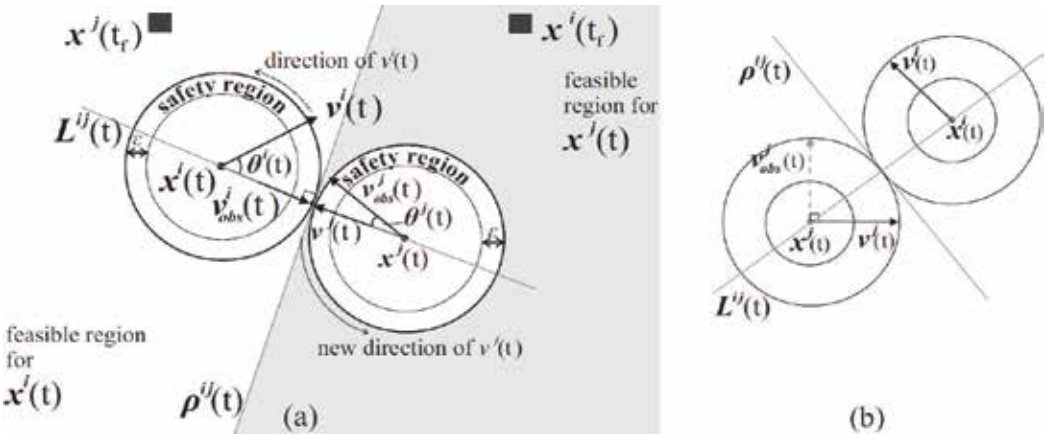

Figure 12. The effects of cross-path conflict resolution.

moved to obtain smoother phase transition is to measure $\theta^i(t)$ and $\theta^j(t)$. If $\theta^j(t) < \theta^i(t)$, then $v^j(t)$ should be moved. This is done by swapping $v^j(t)$ and $v^j_{obs}(t)$, which immediately results in a simple collision reconfiguration. Thereafter, when the Q-CAC algorithm expands $\theta^j(t)$, it is the former $v^j_{obs}(t)$ (which is now the new $v^j(t)$) that moves, whereas the former $v^j(t)$ (which is now the new $v^j_{obs}(t)$) remains static.

Therefore, if a cross-path trajectory is determined, to resolve the problem it is sufficient to swap the variables in one of the avoidance constraints (Eq. (21) or Eq. (22)). For example, Eq. (21) may be left as it is and Eq. (22) is rewritten in the form

$$\begin{bmatrix} 2(\cos\varnothing + \mu) & \begin{bmatrix} v^j_{obs}(k+2) \\ v^j(k+2) \end{bmatrix}^T \\ \begin{bmatrix} v^j_{obs}(k+2) \\ v^j(k+2) \end{bmatrix} & M \end{bmatrix} \geq 0. \tag{28}$$

The trajectories obtained by applying this strategy to cross-path collision avoidance for two vehicles in 2D and 3D are shown in [14].

3.3.5. Resolving cross-path collision for more than two vehicles

If more than two vehicles are involved as shown in **Figure 13**, for any vehicle i, whose attitude vector $v^i(t)$ is in a cross-path configuration with vehicles j and k, we are concerned only about the two bounding obstacle vectors $v^{ij}_{obs}(t)$ and $v^{ik}_{obs}(t)$.

In order not to get into a stalemate situation (undesirable for aircraft), only positive nonzero velocities are required to be generated. We adopt a counterclockwise avoidance measure to achieve this, where, for each vehicle, the left bounding obstacle vector is always chosen as the

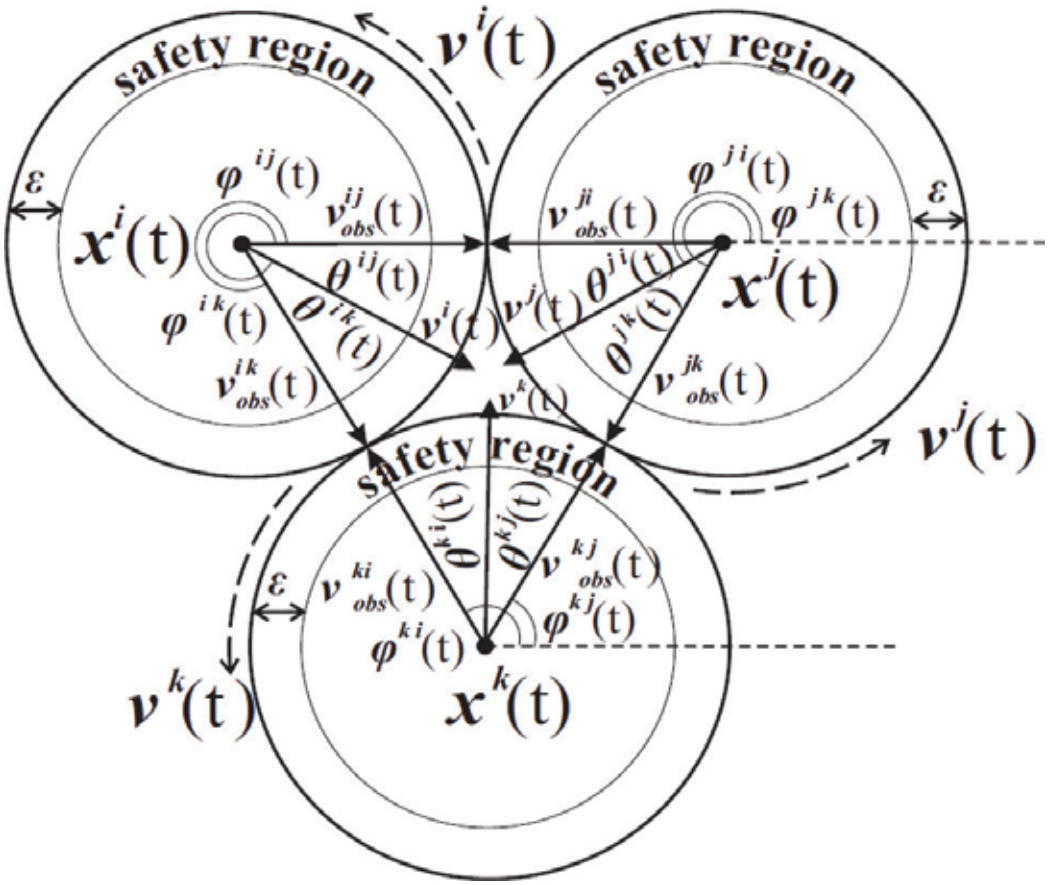

Figure 13. Three-vehicle cross-path trajectory problem.

cross-path obstacle vector for avoidance. For example, for k to turn counterclockwise, it chooses the vector $v^{ki}_{obs}(t)$ to avoid instead of $v^{kj}_{obs}(t)$. Thus, for the configuration of **Figure 13**, the following set of attitude constraints is enforced:

$$\begin{bmatrix} 2(\cos\varnothing+\mu) & \begin{bmatrix} v^{ij}_{obs}(k+2) \\ v^{i}(k+2) \end{bmatrix}^T \\ \begin{bmatrix} v^{ij}_{obs}(k+2) \\ v^{i}(k+2) \end{bmatrix} & M \end{bmatrix} \geq 0, \tag{29}$$

$$\begin{bmatrix} 2(\cos\varnothing+\mu) & \begin{bmatrix} v^{jk}_{obs}(k+2) \\ v^{j}(k+2) \end{bmatrix}^T \\ \begin{bmatrix} v^{jk}_{obs}(k+2) \\ v^{j}(k+2) \end{bmatrix} & M \end{bmatrix} \geq 0, \tag{30}$$

$$\begin{bmatrix} 2(\cos\varnothing+\mu) & \begin{bmatrix} v_{obs}^{ki}(k+2) \\ v^{k}(k+2) \end{bmatrix}^{T} \\ \begin{bmatrix} v_{obs}^{ki}(k+2) \\ v^{k}(k+2) \end{bmatrix} & M \end{bmatrix} \geq 0. \tag{31}$$

3.4. Consensus with Q-CAC–based avoidance

Once a safe attitude vector $v^i(k)$ is computed at time k for any i, the next position $x^i(k+1)$ is computed as a point a distance $r^{*i}/2$ from the current position, along the vector $v^i(k)$. Note that $v^i(k)$ is normalized to keep the computed control bounded. Whether there are intersections of the safety regions or not, one can guarantee the safety of the algorithm by bounding the control size within the interval $0<u^i\leq r^{*i}/2$. This means that a vehicle never steps beyond its safety region at any single time step.

Another important consideration is the size of control computed at each time using Laplacian matrices, which is directly proportional to the algebraic connectivity of the communication graph, and inversely proportional to the magnitude of the current time k. This means that, while the early values of $\mathbf{u}$ are large and therefore unsafe for collision avoidance (and must be bounded), the latter values of $\mathbf{u}$ are very small and therefore slow down the rate of convergence. One can observe that collisions are less likely to occur in the latter times when the vehicles are closer to their goal positions; consequently, convergence is slower at that time. Therefore, there is need to obtain constantly bounded control $\mathbf{u}$ which can guarantee both collision avoidance and a high speed of convergence. The following modifications to Eq. (4) and Eq. (5) were proposed in our previous works [5, 9, 14]. For the leader-follower architecture,

$$\mathbf{u} = -\eta \log_{10}(k+1)\frac{\Delta t}{2\lambda_2(\boldsymbol{L})}\Gamma\left(\boldsymbol{x}-\boldsymbol{x}^{\boldsymbol{off}}\right) - \beta \log_{10}(k+1)\frac{\Delta t}{2\lambda_2(\boldsymbol{L})}\mathbf{K}\left(\boldsymbol{x}-\boldsymbol{x}^{\boldsymbol{off}}\right). \tag{32}$$

And for the leaderless architecture,

$$\mathbf{u} = -\eta \log_{10}(\mathrm{k}+1)\frac{\Delta t}{2\lambda_2(\mathbf{L})}\Gamma\left(\boldsymbol{x}-\boldsymbol{x}^{\boldsymbol{off}}\right) - \beta \log_{10}(\mathrm{k}+1)\frac{\Delta t}{2\lambda_2(\mathbf{L})}\mathbf{K}(\boldsymbol{x}-\boldsymbol{x}_d), \tag{33}$$

where $\lambda_2(\boldsymbol{L})$ is the second smallest eigenvalue of the Laplacian $\mathbf{L}$. The parameter η is for scaling the consensus term and β is for scaling the proportional term in Eqs. (32) and (33). The logarithmic term $\log_{10}(k+1)$ and the term $\frac{\Delta t}{2\lambda_2(\boldsymbol{L})}$ are used to reduce $\|\boldsymbol{u}\|$ when k is small and increase $\|\boldsymbol{u}\|$ when k is large. The choices of parameters η and β should depend on the radius of $\mathbf{S}$ and safety region ε for each vehicle. Alternatively, one may choose to compute an unbounded $\mathbf{u}$ using Eqs. (4) or (5), then for each $u^i > \frac{r^{*i}}{2}$, normalize u^i and set $u^i = \frac{r^{*i}}{2}u^i$.

The step-by-step procedure for implementing the algorithm including a flowchart can be found in Ref. [14].

4. Simulation results

To demonstrate the solutions developed in this chapter, we revisit the experiment presented in **Figure 1**. The robots are homogeneous, and **S** for each robot is 85 mm, $\varepsilon = 90$ mm, whereas the dimensions of the soccer pitch are 6050 mm x 4050 mm. In **Figure 14** (a), Eq. (5) was applied with the cyclic communication topology with one leader (**Figure 2**). In **Figure 14** (b), Eq. (33) was applied with a full communication topology (i.e., every vehicle can communicate with each other). The simulation was done with MATLAB R2009a on an Intel® Core(TM)2 Duo P8600 @ 2.40 GHz with 2 GB RAM, running Windows 7. For **Figure 14(a)**, the multipath planning problem took 244 time-steps to solve, resulting in a total computation time of 7.343 s, in which 203 avoidance attempts were made, and there were no collisions. For **Figure 14(b)**, using a full communication topology, the computational time was 0.0131 s, and there were no collisions.

In [14], more simulations and analyses are presented, together with the limitations of this approach, which remains to be explored for future development.

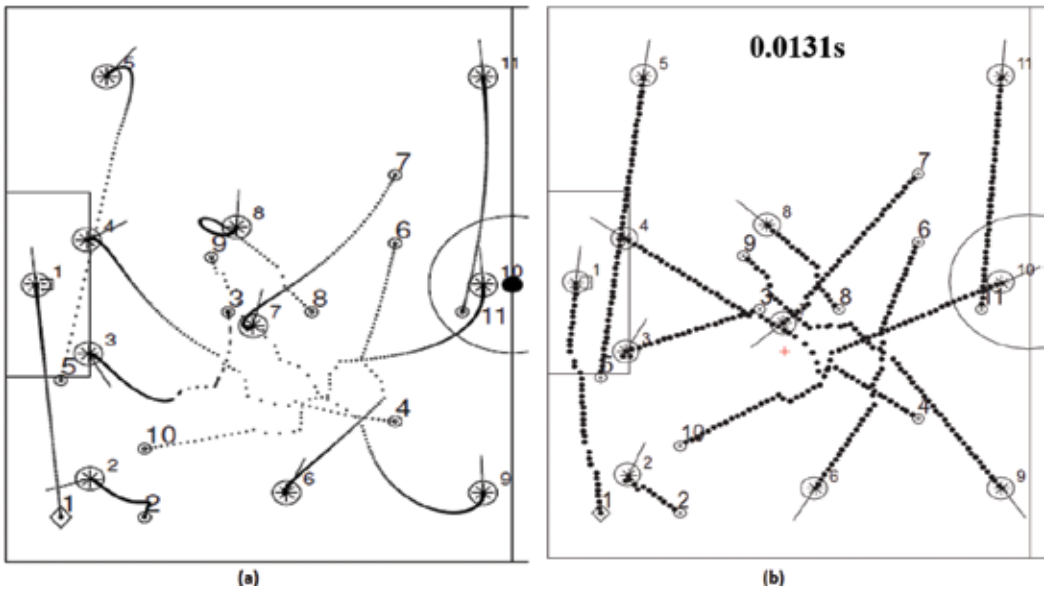

Figure 14. Collision-free reconfiguration: (a) using topology with Eq. (5) and (b) using fully connected graph with Eq. (33).

5. Conclusion

In this chapter, we considered consensus-based multipath planning. An approach to incorporating collision avoidance in adversarial situations in the consensus algorithm by applying Q-CAC is presented. Simulation results are presented here to show that for a sizable number of

vehicles, collision avoidance and fast convergence are guaranteed. Future work will include implementation on a team of mobile robots and autonomous aerial vehicles.

Author details

Innocent Okoloko

Address all correspondence to: okoloko@ieee.org

Department of Electrical Engineering, Universidad de Ingenieria y Tecnologia, Lima, Peru

References

[1] Zickler S, Laue T, Birbach O, Wongphati M, Veloso M. SSL-vision: The shared vision system for the RoboCup small size league. In: RoboCup 2009: Robot Soccer World Cup XIII. Baltes J, Lagoudakis MG, Naruse T, Shiry S, editors. Lecture Notes in Artificial Intelligence. Springer; 2010. p. 425-436. ISBN 978-3-642-11876-0

[2] Waymo. Google's Self-Driving Cars [Internet]. 2016. Available from: https://waymo.com/ [Accessed: August, 2017]

[3] PATH. California Partners for Advanced Transit and Highways [Internet]. 2006. Available from: http://www.path.berkeley.edu/ [Accessed: December, 2009]

[4] Blackwood G, Lay O, Deininger B, Gudim M, Ahmed A, Duren R, Noeckerb C, Barden B. The StarLight mission: A formation-flying stellar interferometer. In: SPIE 4852, Interferometry in Space; 22 August; Waikoloa, Hawaii. SPIE Digital Library; 2002. DOI: http://dx.doi.org/10.1117/12.460942

[5] Okoloko I, Basson A. Consensus with collision avoidance: An LMI approach. In: 5th IEEE International Conference on Automation, Robotics and Applications; 06–08 December; Wellington, NZ. IEEE; 2011. ISBN:9781457703287

[6] Chandler PR, Pachter M, Rasmussen S. UAV cooperative control. In: IEEE ACC; 25–27 June; Arlington, VA. IEEExplore; 2001. p. 50-55. DOI: 10.1109/ACC.2001.945512

[7] Richards A, Schouwenaars T, How JP, Feron E. Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming. AIAA Journal of Guidance Control and Dynamics. 2002;**25**:755-764. DOI: https://doi.org/10.2514/2.4943

[8] Okoloko I. Consensus based distributed motion planning on a sphere. In: IEEE ACC; 17–19 June; Washington DC. IEEEXplore; 2013. p. 6132-6137. DOI: 10.1109/ACC.2013.6580799

[9] Okoloko I. Path planning for multiple spacecraft using consensus with LMI avoidance constraints. In: IEEE Aerospace Conference; 3–10 March; Big Sky, Montana. IEEEXplore; 2012. p. 1-8. DOI:10.1109/AERO.2012.6187118

[10] Okoloko I, Kim Y. Distributed constrained attitude and position control using graph Laplacians. In: ASME Dynamic Systems and Control Conference; 13–15 September; Cambridge, Massachusetts. ASME; 2010. p. 377-383. DOI: 10.1115/DSCC2010-4036

[11] Hwang I, Tomlin C. Protocol-based conflict resolution for finite information horizon. In: IEEE ACC; 8–10 May; Anchorage, Alaska. IEEExplore; 2002. p. 748-753. DOI: 10.1109/ACC.2002.1024903

[12] Peng L, Zhao Y, Tian B, Zhang J, Bing-Hong W, Hai-Tao Z, Zhou T. Consensus of self-driven agents with avoidance of collisions. Physical Review. 2009;**79**(E). DOI: 10.1103/PhysRevE.79.026113

[13] Olfati-Saber R. Flocking for multi-agent dynamic systems: Algorithms and theory. IEEE Transactions on Automatic Control. 2006;**51**(3):401-420. DOI: 10.1109/TAC.2005.864190

[14] Okoloko I. Multi-Path Planning and Multi-Body Constrained Attitude Control [Dissertation]. Stellenbosch, South Africa: PhD Thesis: Stellenbosch University; 2012. p. 185. Available from: http://hdl handle.net/10019.1/71905

[15] Fax AJ. Optimal and Cooperative Control of Vehicle Formations [Thesis]. Pasadena, CA: PhD Thesis, CALTECH; 2002. p. 135. Available from: thesis.library.caltech.edu/4230/1/Fax_ja_2002.pdf

[16] Kim Y, Mesbahi M. Quadratically constrained attitude control via semidefinite programming. IEEE Transactions on Automatic Control. 2004;**49**:731-735. DOI: 10.1109/TAC.2004.825959

[17] Boyd S, Ghaoui LE, Feron E, Balakrishnan V. Linear Matrix Inequalities in System and Control Theory. Philadelphia, PA: SIAM; 1994. p. 205. ISBN: 0-89871-334-X

Chapter 2

Multi-Path Planning on a Sphere with LMI-Based Collision Avoidance

Innocent Okoloko

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.71216

Abstract

The problem of path planning with collision avoidance for autonomous flying vehicles will become more critical as the density of such vehicles increase in the skies. Global aerial navigation paths can be modeled as a path-planning problem on a unit sphere. In this work, we apply consensus theory and semidefinite programming to constrained multi-path planning with collision avoidance for a team of communicating vehicles navigating on a sphere. Based on their communication graph, each vehicle individually synthesizes a time-varying *Laplacian-like* matrix which drives each of them from their initial positions to consensus positions on the surface of the sphere. The solution trajectories obtained on the unit sphere are transformed back to actual vehicle coordinates. Formation configurations are realized via consensus theory, while collision avoidance is realized via semidefinite programming. A Lyapunov-based stability analysis is also provided, together with simulation results to demonstrate the effectiveness of the approach.

Keywords: consensus, path planning, avoidance, optimization, LMI

1. Introduction

In this chapter, we present an approach to constrained multi-agent control on the unit sphere; by applying consensus theory and constrained attitude control (CAC) via semidefinite programming. Global navigation can be modeled by control on the unit sphere and such algorithms have applications in: aerial navigation [1]; sea navigation and ocean sampling [2]; space navigation and satellite cluster positioning [3, 4]. For example, the algorithm presented in this chapter will find practical application in *aircraft horizontal separation*.

Most path-planning work generally focus on two-dimensional (2D) [5, 6], and three- dimensional motion planning (3D) [7–10]. However, both path planning models are limited when the motion is constrained to evolve on a sphere.

Looking at the main works that have been done on control on a sphere, [11] applied Lie algebra to develop a model of self-propelled particles (as point masses) which move on the surface of a unit sphere at constant speed. Circular formations of steady motions of the particles around a fixed small circle on the sphere were identified as relative equilibria of the model using a Lie group representation. The paper also provided mathematically justified shape control laws that stabilize the set of circular formations. They also proposed a shape control to isolate circular formations of particles with symmetric spacing by using *Laplacian control*. Further work on this is presented in [12].

The works [11, 12] are based on [5, 13], where a geometric approach to the gyroscopic control of vehicle motion in planar and three-dimensional particle models was developed for formation acquisition and control with collision avoidance in *free space*. They discovered three possible types of relative equilibria for their unconstrained gyroscopic control system on $SE(3)$: (i) parallel particle motion with arbitrary spacing; (ii) circular particle motion that has a common radius, axis and direction of rotation, and arbitrary along-axis spacing; (iii) helical particle motion that has a common radius, axis and direction of rotation, along-axis speed (pitch) and arbitrary along-axis spacing. This approach is effective in formation control of multiple systems in *unconstrained spaces* and for formations that conform to the three types of relative equilibria described above.

Therefore, it is necessary to consider consensus on a sphere, which can be applied to the more general motion control problem involving: (i) constrained spaces which contain static obstacles such as clutter; (ii) speed constrained vehicles; (iii) other arbitrary formations which are different from the relative equilibria described above. We apply consensus theory to collective motion of a team of communicating vehicles on the sphere and the concept of constrained attitude control (CAC) to generate collision avoidance behavior among the vehicles as they navigate to arbitrary formations [14].

We assume that each individual vehicle can communicate with *neighbors* within its sensor view. Each vehicle can therefore use the *Laplacian matrix* of the communication graph $\mathbf{L}$ in a semidefinite program to plan consensus trajectories on the sphere. Then the concept of CAC is used to incorporate collision avoidance, by maintaining specified minimum angles between vectors of vehicle positions. The algorithm presented here is applicable to motion control in both *constrained* and *unconstrained* spaces on the sphere, e.g. for planning consensus trajectories around static obstacles or adversarial non-cooperative obstacles on the sphere. The approach can also be applied to constrained vehicle motion of non-constant velocities. It is also possible to generate formations on the sphere that are different from circular motion.

The rest of this chapter is organized as follows. In Section 2, we present the mathematical basis of consensus theory, while the problem statement is presented Section 3. In Section 4, the solution and convergence analysis are presented. This is followed by simulation results in Section 5 and references in Section 6. **Table 1** lists frequently used notation in this chapter.

Notation	Meaning
n	Number of vehicles
i	Vehicle number i
x^i	Position vector of vehicle i
$\widehat{x}$	Unit vector corresponding to vector x
$u^i, \dot{x}^i$	Control input of vehicle i
x^j_{obs}	Obstacle vector number j
$\left(x^{ij}\right)^{off}$	Offset vector between vehicles i and j
φ^{ij}	Angle between vehicle i and obstacle j
θ^{ij}	Angle between vehicles i and j
α^i	Minimum angular separation from obstacle number i
β^{ij}	Minimum angular separation between vehicles i and j
$\mathbf{x}$	Stacked vector of n position vectors
$\mathbf{u}, \dot{\mathbf{x}}$	Stacked vector of n control inputs
$\mathbf{L}$	Laplacian matrix, $\mathbf{L} = \mathbf{D}_{\mathcal{G}} - \mathbf{A}_{\mathcal{G}}$
$\mathcal{L}, \mathcal{L}^i$	Laplacian-like stochastic matrix
$\mathbf{0}$	A vector consisting of all zeros
$\otimes$	Kronecker multiplication operator
$SE(3)$	Special Euclidean group
$\mathcal{S}^m$	The set of $m \times m$ positive definite matrices
$\mathbf{I}_n$	The $n \times n$ identity matrix
Λ	A positive definite matrix variable, $\Lambda \in S^m$
$\mathcal{C}$	The consensus space for $\mathbf{x}$, $\mathcal{C} = \left\{\mathbf{x} \middle\vert x^1 = x^2 =, \cdots, = x^n\right\}$
$\mathcal{G}$	Graph
$\mathcal{V}$	Set of vertices of $\mathcal{G}$
$\mathcal{E}$	Set of edges of $\mathcal{G}$
v_i	Vertex $v_i \in \mathcal{V}$
(v_i, v_j)	Endpoint or edge $(v_i, v_j) \in \mathcal{E}$
$\mathcal{N}_i$	Neighbors of v_i; $\mathcal{N}_i = \left\{v_j \in \mathcal{V} : (v_i, v_j) \in \mathcal{E}\right\}$
$\mathbf{A}_{\mathcal{G}}$	Adjacency matrix of $\mathcal{G}$; $\mathbf{A}_{\mathcal{G}} = \left[a_{ij}\right]$
$\mathbf{D}_{\mathcal{G}}$	Out-degree matrix of $\mathcal{G}$; $\mathbf{D}_{\mathcal{G}} = \left[d_{ij}\right]$
$\mathbf{S}$	A vector or matrix in the Schur's inequality
$\mathbf{R}$	A positive definite matrix in the Schur's inequality
$\mathbf{Q}$	A symmetric matrix in the Schur's inequality
$\mathbf{M}$	A positive definite matrix variable
$\mathbf{G}$	A positive semidefinite matrix

Table 1. Frequently used notation in this chapter.

2. Mathematical background

This section briefly describes the mathematical basis of consensus theory.

2.1. Basic graph theory

We define a graph $\mathcal{G}$ as a pair $(\mathcal{V}, \mathcal{E})$ consisting of two finite sets having elements; a set of points called *vertices* $\mathcal{V} = \{1, 2, \cdots, n\}$, and a set of connecting lines called *edges*, $\mathcal{E} \subseteq \{(v_i, v_j) : v_i, v_j \in \mathcal{V}, j \neq i\}$ or *endpoints*, $\mathcal{E}(i, j)$ or (v_i, v_j), of the vertices [15]. Thus, an edge is *incident* with vertices v_i and v_j. Graph $\mathcal{G}$ is said to be *undirected* if for every edge connecting two vertices, communication between the vertices is possible in both directions across the edge, i.e. $(v_i, v_j) \in \mathcal{E}$ implies $(v_j, v_i) \in \mathcal{E}$; otherwise it is called a *directed graph* (*digraph*), and it is *symmetric*. The quantity $|\mathcal{V}|$ is called the *order*, and $|\mathcal{E}|$ the *size*, respectively, of $\mathcal{G}$. The set of *neighbors* of node v_i is denoted by $\mathcal{N}_i = \{v_j \in \mathcal{V} : (v_i, v_j) \in \mathcal{E}\}$. The number of edges **incident with** vertex v is called the *degree* or *valence* of v. Furthermore, the number of *directed* edges **incident into** v is called the *In-degree* of v, while the *Out-degree* is similarly defined as the number of edges **incident out** of the v.

We define the *adjacency matrix* $\mathbf{A}_\mathcal{G} = [a_{ij}]$ of $\mathcal{G}$ of order n as the $n \times n$ matrix

$$a_{ij} = \begin{cases} 1 & \text{if } e(i,j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

For the undirected graph $\mathbf{A}_\mathcal{G}$ is *always symmetric*, while $\mathbf{A}_\mathcal{G}$ of a digraph G is symmetric if and only if $\mathcal{G}$ is symmetric. The *out-degree matrix* $\mathbf{D}_\mathcal{G} = [d_{ij}]$ of $\mathcal{G}$ of order n, is an $n \times n$ matrix

$$d_{ii} = \sum_{i \neq j} a_{ij}, \tag{2}$$

which is simply the diagonal matrix with each diagonal element equal to the out-degree of the corresponding vertex. The *in-degree matrix* of $\mathcal{G}$ is similarly defined.

The *Laplacian matrix* $\mathbf{L} = [l_{ij}]$ of digraph $\mathcal{G}$ of order n, is the $n \times n$ matrix

$$\mathbf{L} = \mathbf{D}_\mathcal{G} - \mathbf{A}_\mathcal{G} \tag{3}$$

An important property of any Laplacian $\mathbf{L}$ is that its rows and columns, sum to zero.

2.2. Basic consensus theory

The basic *consensus* problem is that of driving the states of a team of communicating agents to an agreed state, using distributed protocols based on their *communication graph*. In this framework, the agents (or vehicles) $i (i = 1, \cdots, n)$ are represented by vertices of the graph, while the edges of the graph represent communication links between them. Let x^i denote the state of a vehicle i and $\mathbf{x}$ is the stacked vector of the states all vehicles in the team. For systems modeled

by first-order dynamics, the following first-order consensus protocol (or its variants) has been proposed, e.g. [16, 17]

$$\dot{\mathbf{x}}(t) = -\mathbf{L}\left(\mathbf{x}(t) - \mathbf{x}^{off}\right). \tag{4}$$

We determine that consensus has been achieved when $\left\|x^i - x^j\right\| \to \left(x^{ij}\right)^{off}$ as $t \to \infty$, $\forall\, i \neq j$. A more comprehensive presentation of the necessary mathematical tools for this work (including graph theory and consensus theory), can be found in [18].

3. Problem statement

We state the problem of constrained motion on a unit sphere as follows: given a set of communicating vehicles randomly positioned on a unit sphere, with initial positions $x^i(t_0) \in \mathbb{R}^3$, $i = 1, \cdots, n$ (referenced to a coordinate frame centered on the centroid of the sphere), a set of obstacles $x^j_{obs} \in \mathbb{R}^3$, $j = 1, \cdots, m$, and the Laplacian matrix of their communication graph $\mathbf{L}$, find a sequence of collision-free consensus trajectories along the surface of the unit sphere. In this development, a vehicle is modeled as a point mass.

The problem is illustrated in **Figure 1**; the unit sphere is centered on $\mathbf{0}$ which implies that vectors x^i and x^j_{obs} are unit vectors and must be kept so throughout the evolution of the trajectory vectors. The angle between the position vectors of vehicles i and j is θ^{ij}, while φ^{ik} is

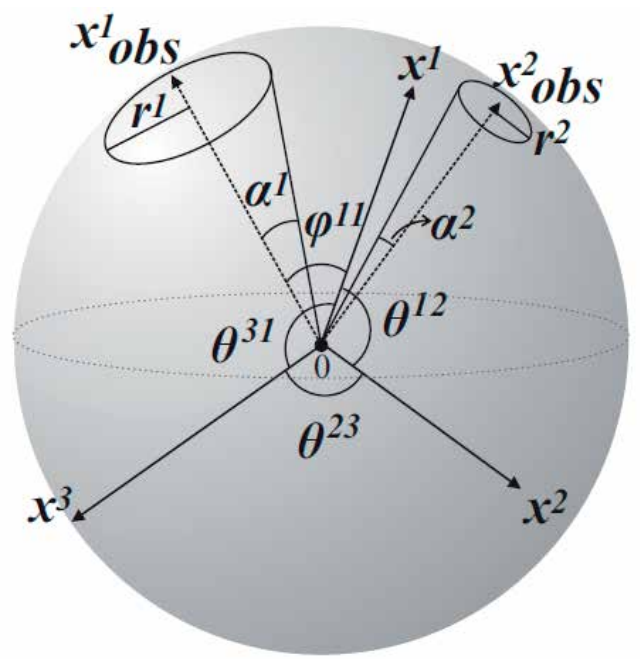

Figure 1. Constrained position control on a unit sphere.

the angle between vehicle i and obstacle k. The control problem is to drive all x^i to a consensus position or to a formation while avoiding each other and the x^j_{obs} along the way on the unit sphere. From the solution trajectories, obtained as unit vectors, the actual desired vehicle trajectories are recovered via scalar multiplication and coordinate transformation.

There are two parts to the problem: *consensus* and *collision avoidance*. The consensus part is that of incorporating consensus behavior into the team on the unit sphere, which can lead to collective motion such as *rendezvous*, *platooning*, *swarming* and other *formations*. The second part which is *collision avoidance*, is resolved by applying constrained attitude control (CAC). The solutions are presented in the next section.

4. Solutions

We develop a solution that incorporates four steps: (i) synthesis of position consensus on the unit sphere; (ii) formulation of CAC based collision avoidance on the unit sphere; (iii) formulation of formation control on the unit sphere; (iv) consensus-based collision-free arbitrary reconfigurations on the unit sphere.

4.1. Synthesis of position consensus on the unit sphere

The basic consensus protocol Eq. (4) on its own does not solve the consensus problem on a sphere; neither does it solve the collision avoidance problem in *adversarial situations* (when there is opposing motion and static obstacles). To incorporate consensus on a unit sphere, we follow an optimization approach, by coding requirements as a set of linear matrix inequalities (LMI) and solving for consensus trajectories on the sphere. The main problem at this stage is to find a *feasible sequence of consensus trajectories* for each vehicle on the sphere, which satisfies *norm* and avoidance constraints. For this purpose, rather than state the objective function as a minimization or maximization problem (as usual in optimization problems), we state the objective function as the discrete time version of a *semidefinite consensus dynamics*, which will be augmented with an arbitrary number of constraints.

A basic requirement is that any vehicle i can communicate with at least one other neighboring vehicle. Given that τ is the number of vehicles in the neighborhood of i that it can communicate with, then i, $(i = 1, \cdots, n)$ individually synthesizes a Laplacian-like *stochastic matrix* $\mathcal{L}^i$ so that all x^i are driven to consensus on the unit sphere. The synthesis of $\mathcal{L}^i$ is as follows. A semidefinite matrix variable, $\Lambda^i \in \mathbb{S}^3$ for each i is generated. Then

$$\begin{aligned} \mathcal{L}^i(t) &= \left[\tau\Lambda^i_1(t) - \Lambda^i_2(t) \ \cdots \ \Lambda^i_\tau(t)\right], \\ \dot{x}^i(t) &= \left[\tau\Lambda^i_1(t) - \Lambda^i_2(t) \ \cdots \ \Lambda^i_\tau(t)\right]\left[x^T_1(t)\, x^T_2(t) \ \cdots \ x^T_\tau(t)\right]^T \\ &= -\mathcal{L}^i(t)\left[x^T_1(t)\, x^T_2(t) \ \cdots \ x^T_\tau(t)\right]^T, \end{aligned} \tag{5}$$

where $x^T_i(t), i = 1, \cdots, \tau$ are the position vectors of vehicles that i is communicating with at time t. For the purpose of analysis, the collective description for n vehicles is given as

$$\mathcal{L}(t) = \underbrace{\begin{bmatrix} \Lambda^1(t) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \Lambda^n(t) \end{bmatrix}}_{\Lambda(t)} \underbrace{\begin{bmatrix} l_{11}\mathbf{I}_3 & \cdots & l_{1n}\mathbf{I}_3 \\ \vdots & \ddots & \vdots \\ l_{n1}\mathbf{I}_3 & \cdots & l_{nn}\mathbf{I}_3 \end{bmatrix}}_{\Gamma = \mathbf{L} \otimes \mathbf{I}_3}, \tag{6}$$

where, $\mathbf{L} = \left[l_{ij}\right]$, $(i, j = 1, \cdots, n)$ is the collective Laplacian matrix. Note that any Λ^i is unknown, we only want it to be positive semidefinite, therefore it is an optimization variable.

We can now define a collective semidefinite consensus protocol on a sphere as

$$\dot{\mathbf{x}}(t) = -\mathcal{L}(t)\mathbf{x}(t). \tag{7}$$

The Euler's first-order discrete time equivalents of Eqs. (5) and (7) are

$$x^i_{k+1} = x^i_k - \Delta t \mathcal{L}^i(t) x^i_k, \tag{8}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \Delta t \dot{\mathbf{x}}_k = \mathbf{x}_k - \Delta t \mathcal{L}(t)\mathbf{x}_k \tag{9}$$

Each vehicle builds a SDP in which Eq. (8) is included as the dynamics constraint, augmented with several required convex constraints. For example, for the solution trajectories to remain on the unit sphere, norm constraints will be defined for each i as

$$\left(x^i\right)^T_k \left(x^i_{k+1} - x^i_k\right) = 0. \tag{10}$$

Eq. (10) is the discrete time version of $x^i(t)^T \dot{x}^i(t) = 0$ or $\mathbf{x}(t)^T \dot{\mathbf{x}}(t) = 0$, which guarantees that $x^i(t)^T x^i(t) = 1$ or $\mathbf{x}(t)^T \mathbf{x}(t) = n$ for n vehicles, iff $\left\|x^i(0)\right\| = 1 \forall\, i$. Eq. (8) drives the positions $x^i(0)$ to consensus while the norm constraint Eq. (10) keeps the trajectories on the unit sphere.

Theorem 1: As long as the associated (static) communication graph of **L** has a spanning tree, the strategy $\dot{\mathbf{x}}(t) = -\mathbf{L}\mathbf{x}(t)$ achieves global consensus asymptotically for **L** [19].

Proof: The proof [19], is essentially that of convergence of the first-order consensus dynamics.

Next, we use the proof of Theorem 1 as a basis to develop the proof convergence of Eq. (7).

Theorem 2: The time varying system Eq. (7) achieves consensus if **L** is connected. Note that this proof had already been presented in [20].

Proof: Note that if x belongs to the *consensus space* $\mathcal{C} = \left\{\mathbf{x} | x^1 = x^2 =, \cdots, = x^n\right\}$, then $\dot{\mathbf{x}} = \mathbf{0}$, (i.e. all vehicles have stopped moving). Because $\mathcal{C}$ is the nullspace of $\mathcal{L}(t)$, where $\mathcal{L}(t)\mathbf{x} = 0\ \forall \mathbf{x}$. Meaning that once **x** enters $\mathcal{C}$ it stays there since there is no more motion. If consensus has not been achieved then $\mathbf{x} \notin \mathcal{C}$, consider a Lyapunov candidate function $\mathrm{V} = \mathbf{x}^T \Gamma \mathbf{x}$; $\mathrm{V} > 0$ unless $\mathbf{x} \in \mathcal{C}$. Then,

$$\begin{aligned} \dot{V} &= \mathbf{x}^T \Gamma \dot{\mathbf{x}} + \dot{\mathbf{x}}^T \Gamma \mathbf{x}, \\ &= -\mathbf{x}^T \Gamma \mathcal{L}(t) \mathbf{x} - \mathbf{x}^T \mathcal{L}(t)^T \Gamma \mathbf{x}, \\ &= -\mathbf{x}^T \Gamma \Lambda(t) \Gamma \mathbf{x} - \mathbf{x}^T \Gamma \Lambda(t) \Gamma \mathbf{x}, \\ &= -2\mathbf{x}^T \Gamma \Lambda(t) \Gamma \mathbf{x}, \\ &= -2z^T \Lambda(t) z, \end{aligned} \tag{11}$$

where $z = \Gamma \mathbf{x} \neq 0$ for $\mathbf{x} \notin \mathcal{C}$. This implies that $\mathbf{x}$ approaches a point in $\mathcal{C}$ as $t \to \infty$, which proves the claim. Eq. (11) is true for as long as $\mathbf{L}$ is nonempty, i.e., if some vehicles can sense, see or communicate with each other at all times.

4.2. Formulation of CAC based collision avoidance on the unit sphere

To incorporate collision avoidance, we apply the concept of constrained attitude control (CAC), as illustrated in **Figure 1**. We want the time evolution of the position vectors $x^1(t)$, $x^2(t)$ and $x^3(t)$ to avoid two constraint regions around x^1_{obs} and x^2_{obs}. The obstacle regions are defined by cones, whose base radii are r^1 and r^2, respectively. Let the angle between vehicles i and j be θ^{ij}, and that between vehicle i and obstacle k be φ^{ik}. Then the requirements for collision avoidance are: $\varphi^{11} \geq \alpha^1$, $\varphi^{21} \geq \alpha^1$, $\varphi^{31} \geq \alpha^1$, and $\varphi^{12} \geq \alpha^2$, $\varphi^{22} \geq \alpha^2$, $\varphi^{32} \geq \alpha^2$, $\forall t \in [t_0, t_f]$. They have the following equivalent quadratic constraints:

$$x^1(t)^T x^1_{obs} \leq \cos \alpha^1, \tag{12}$$

$$x^2(t)^T x^1_{obs} \leq \cos \alpha^1, \tag{13}$$

$$x^3(t)^T x^1_{obs} \leq \cos \alpha^1, \tag{14}$$

$$x^1(t)^T x^2_{obs} \leq \cos \alpha^2, \tag{15}$$

$$x^2(t)^T x^2_{obs} \leq \cos \alpha^2, \tag{16}$$

$$x^3(t)^T x^2_{obs} \leq \cos \alpha^2. \tag{17}$$

By using the *Schur's complement formula* [21], the above constraints will be converted to the form of linear matrix inequalities (LMI) in order to include them into the respective SDPs. The Schur's complement formula states that the inequality

$$\mathbf{S}\mathbf{R}^{-1}\mathbf{S}^{\mathrm{T}} - \mathbf{Q} \leq 0 \tag{18}$$

where $\mathbf{Q} = \mathbf{Q}^{\mathrm{T}}$, $\mathbf{R} = \mathbf{R}^{\mathrm{T}}$, and $\mathbf{R} > 0$, is equivalent to, and can be represented by the linear matrix inequality

$$\begin{bmatrix} \mathbf{Q} & \mathbf{S} \\ \mathbf{S}^{\mathrm{T}} & \mathbf{R} \end{bmatrix} \geq 0. \tag{19}$$

Next, we attempt to make our quadratic constraints to look like the Schur's inequality. Observe that Eq. (12) is equivalent to

$$\underbrace{\begin{bmatrix} x^1(t)^T & x_{obs}^{1\,T} \end{bmatrix}}_{\mathbf{x^1}(t)^T} \begin{bmatrix} \mathbf{0}_3 & \frac{1}{2}\mathbf{I}_3 \\ \frac{1}{2}\mathbf{I}_3 & \mathbf{0}_3 \end{bmatrix} \underbrace{\begin{bmatrix} x^1(t) \\ x_{obs}^1 \end{bmatrix}}_{\mathbf{x^1}(t)} \leq \cos\alpha^1 \tag{20}$$

Multiply Eq. (20) by 2 and we have

$$\mathbf{x^1}(t)^T \underbrace{\begin{bmatrix} \mathbf{0}_3 & \mathbf{I}_3 \\ \mathbf{I}_3 & \mathbf{0}_3 \end{bmatrix}}_{\mathbf{G}} \mathbf{x^1}(t) \leq 2\cos\alpha^1. \tag{21}$$

We desire a positive definite **G,** i.e. $\mathbf{G} > \mathbf{0}$**,** or whose eigenvalues are all nonnegative (this is synonymous with **R** in the Schur inequality). To make **G** positive definite, one only needs to shift the eigenvalues by choosing a positive real number μ which is larger than the largest absolute value of the eigenvalues of **G**, then

$$\mathbf{x^1}(t)^T \left(\mu\mathbf{I}_6 + \begin{bmatrix} \mathbf{0}_3 & \mathbf{I}_3 \\ \mathbf{I}_3 & \mathbf{0}_3 \end{bmatrix} \right) \mathbf{x^1}(t) \leq 2\left(\cos\alpha^1 + \mu \right). \tag{22}$$

Let $\mathbf{M} = \left(\mu\mathbf{I}_6 + \begin{bmatrix} \mathbf{0}_3 & \mathbf{I}_3 \\ \mathbf{I}_3 & \mathbf{0}_3 \end{bmatrix} \right)^{-1}$, then **M** is positive definite. Thus, following the Schur's complement formula, the LMI equivalent of Eq. (12) becomes

$$\begin{bmatrix} 2\left(\cos\alpha^1 + \mu \right) & \underbrace{\begin{bmatrix} x^1(t) \\ x_{obs}^1 \end{bmatrix}^T}_{} \\ \begin{bmatrix} x^1(t) \\ x_{obs}^1 \end{bmatrix} & \mathbf{M} \end{bmatrix} \geq 0, \tag{23}$$

The LMI equivalents of Eqs. (12) to (17) in discrete time can now be written as follows

$$\begin{bmatrix} 2\left(\cos\alpha^1 + \mu \right) & \underbrace{\begin{bmatrix} x^1(k+1) \\ x_{obs}^1 \end{bmatrix}^T}_{} \\ \underbrace{\begin{bmatrix} x^1(k+1) \\ x_{obs}^1 \end{bmatrix}}_{} & \mathbf{M} \end{bmatrix} \geq 0, \tag{24}$$

$$\begin{bmatrix} 2\left(\cos\alpha^1 + \mu \right) & \underbrace{\begin{bmatrix} x^2(k+1) \\ x_{obs}^1 \end{bmatrix}^T}_{} \\ \underbrace{\begin{bmatrix} x^2(k+1) \\ x_{obs}^1 \end{bmatrix}}_{} & \mathbf{M} \end{bmatrix} \geq 0, \tag{25}$$

$$\begin{bmatrix} 2\left(\cos\alpha^1+\mu\right) & \underbrace{\begin{bmatrix} x^3(k+1) \\ x^1_{obs} \end{bmatrix}^T} \\ \underbrace{\begin{bmatrix} x^3(k+1) \\ x^1_{obs} \end{bmatrix}} & \mathbf{M} \end{bmatrix} \geq 0, \tag{26}$$

$$\begin{bmatrix} 2\left(\cos\alpha^2+\mu\right) & \underbrace{\begin{bmatrix} x^1(k+1) \\ x^2_{obs} \end{bmatrix}^T} \\ \underbrace{\begin{bmatrix} x^1(k+1) \\ x^2_{obs} \end{bmatrix}} & \mathbf{M} \end{bmatrix} \geq 0, \tag{27}$$

$$\begin{bmatrix} 2\left(\cos\alpha^2+\mu\right) & \underbrace{\begin{bmatrix} x^2(k+1) \\ x^2_{obs} \end{bmatrix}^T} \\ \underbrace{\begin{bmatrix} x^2(k+1) \\ x^2_{obs} \end{bmatrix}} & \mathbf{M} \end{bmatrix} \geq 0, \tag{28}$$

$$\begin{bmatrix} 2\left(\cos\alpha^2+\mu\right) & \underbrace{\begin{bmatrix} x^3(k+1) \\ x^2_{obs} \end{bmatrix}^T} \\ \underbrace{\begin{bmatrix} x^3(k+1) \\ x^2_{obs} \end{bmatrix}} & \mathbf{M} \end{bmatrix} \geq 0. \tag{29}$$

Figure 2 shows the result for applying the above strategy to the rendezvous of 4 vehicles on a sphere, with avoidance of a static obstacle x_{obs}, with $\alpha = 30^{\circ}$.

4.3. Formulation of formation control on the unit sphere

Formation patterns are obtained by specifying a minimum angular separation of $\beta^{ij}(t)$ between any two vehicles i and j thereby defining relative spacing between individual vehicles. Using the avoidance strategy formerly described, the constraint $\theta^{ij} \geq \beta^{ij} \forall\, i, j$ is used to define the set of avoidance constraints that will result in the desired formation pattern. The relative spacing results in intervehicle collision avoidance. For n vehicles, the avoidance requirements result in extra $P(n-2) = \frac{n!}{(n-2)!}$ constraints, which are included along with the static obstacle avoidance constraints such as Eqs. (24) to (29). **Figure 3** shows the result for applying the above strategy to the rendezvous with inter-vehicle avoidance and static obstacle avoidance, of four vehicles, using a fully connected graph Topology 1 in **Figure 4**. In this experiment $\alpha = 30^{\circ}$ and the

http://dx.doi.org/10.5772/intechopen.71216

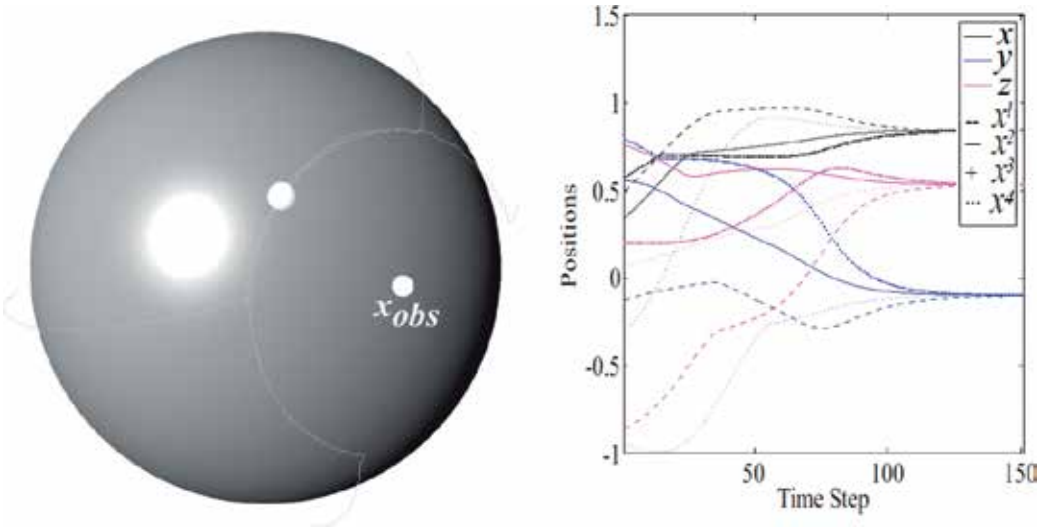

Figure 2. Four-vehicle rendezvous on a unit sphere with collision avoidance of a static obstacle. The figure shows the evolution of the x, y, z positions of the four vehicles x^1, x^1, x^3, x^4 from initial to final positions.

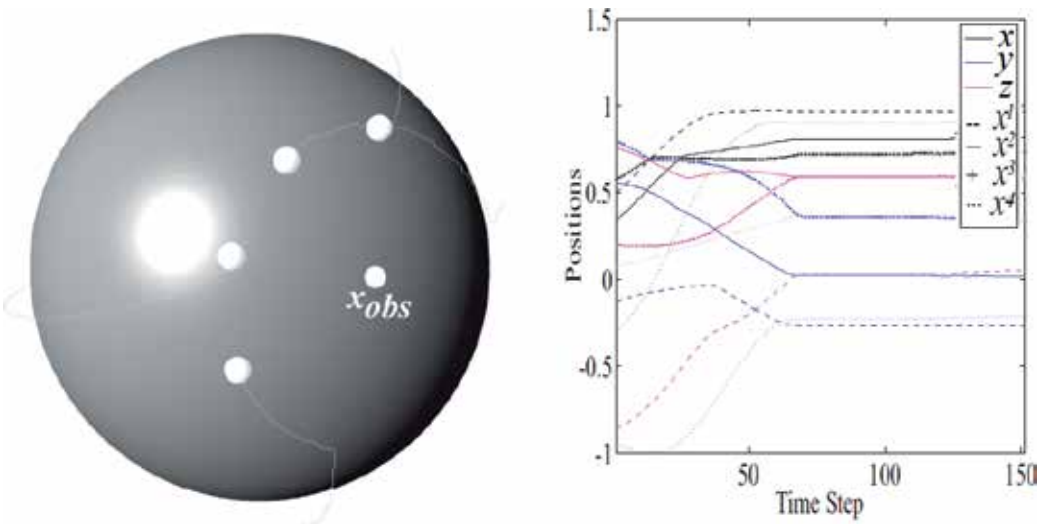

Figure 3. Four-vehicle formation acquisition on a unit sphere with collision avoidance of a static obstacle, and with inter-vehicle collision avoidance. The figure shows the evolution of the x, y, z positions of the four vehicles x^1, x^1, x^3, x^4 from initial to final positions.

minimum angular separations between the vehicles is set at a constant value $\beta^{ij} = 20^o \forall\, i, j$. Therefore, in addition to the four static obstacle avoidance constraints (such as Eqs. (24) to (29), with $\alpha^1 = \alpha$), each vehicle has three more intervehicle collision avoidance constraints such as

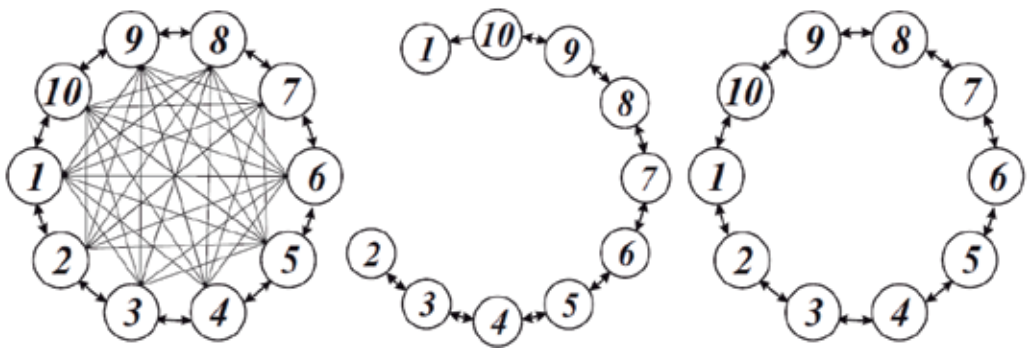

Figure 4. Topology 1 (left) is a *fully connected* communication graph with *no* leader, topology 2 (center) is a *cyclic* communication graph with *one* leader, node 1, and topology 3 (right) is a *cyclic* communication graph with *no* leader.

$$\begin{bmatrix} 2\left(\cos\beta^{ij}+\mu\right) & \underbrace{\begin{bmatrix} x^i(k+1) \\ x^j(k+1) \end{bmatrix}^T} \\ \underbrace{\begin{bmatrix} x^i(k+1) \\ x^j(k+1) \end{bmatrix}} & \mathbf{M} \end{bmatrix} \geq 0, \tag{30}$$

$\forall\, i, j\ (i \neq j)$.

Putting it all together, the optimization problem of finding a feasible sequence of consensus trajectories with collision avoidance on a unit sphere may be posed as a semidefinite program (SDP) as follows. Given the set of initial positions $x^i(t_0)$, $(i = 1 \cdots n)$ and the plant Eq. (5) for each vehicle, find a feasible sequence of trajectories that satisfies the following constraints:

$$x^i_{k+1} = x^i_k - \Delta t \mathcal{L}^i(t) x^i_k, \qquad \text{dynamics constraint}$$

$$\left(x^i\right)^T_k \left(x^i_{k+1} - x^i_k\right) = 0, \qquad \text{norm constraint}$$

$$\begin{bmatrix} 2\left(\cos\alpha^{ij}+\mu\right) & \underbrace{\begin{bmatrix} x^i(k+1) \\ x^j_{obs} \end{bmatrix}^T} \\ \underbrace{\begin{bmatrix} x^i(k+1) \\ x^j_{obs} \end{bmatrix}} & \mathbf{M} \end{bmatrix} \geq 0, \qquad \text{static obstacle avoidance constraint}$$

$$\begin{bmatrix} 2\left(\cos\beta^{ij}+\mu\right) & \underbrace{\begin{bmatrix} x^i(k+1) \\ x^j(k+1) \end{bmatrix}^T} \\ \underbrace{\begin{bmatrix} x^i(k+1) \\ x^j(k+1) \end{bmatrix}} & \mathbf{M} \end{bmatrix} \geq 0, \qquad \text{intervehicle avoidance constraint}$$

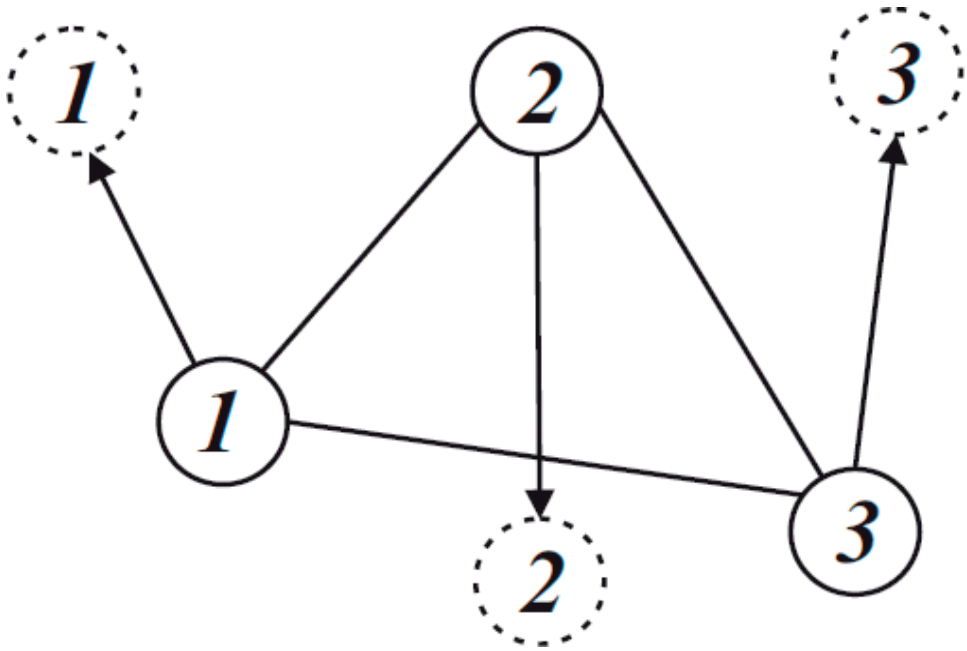

Figure 5. Multiple virtual leaders graph topology with an undirected topology.

where x_{k+1}^i and Λ_k^i (which are components of $\mathcal{L}^i$) are the optimization variables. They are declared as SDP variables where Λ_k^i shapes the trajectories x_{k+1}^i to satisfy norm and avoidance constraints.

4.4. Consensus-based collision-free arbitrary reconfigurations on the unit sphere

Consider a more traditional reconfiguration problem that may not require formation control. For example, in a tracking problem, several vehicles are required to change their positions by tracking that of a set of *virtual leaders*, whose positions may be static or time-varying. For this to be possible, each vehicle must be connected to its corresponding virtual leader via a *leader-follower digraph,* see **Figure 5** for an example topology for three vehicles. In **Figure 5**, the vertices in dashed circles are the states of the virtual leaders, while those with solid circles correspond to the states of the real vehicles. There are three unconnected separate leader follower digraphs (edges indicated with arrows). In addition, there is an *undirected graph* (edges without arrows) which enables the vehicles to communicate bidirectionally to provide data for inter-vehicle collision avoidance.

If $x_v^i(t)$ is the state of a virtual leader corresponding to vehicle i, then for each leader-follower vehicle pair $\mathbf{x}^i(t) = \left[x_v^i(t)^T \; x^i(t)^T\right]^T$ the corresponding leader-follower Laplacian matrix is

$$\mathbf{L}^t(t) = \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix}. \tag{31}$$

The corresponding collective dynamics of $\mathbf{x}^i(t)$ is

$$\dot{\mathbf{x}}^i(t) = -\begin{bmatrix} \Lambda^i(t) & \mathbf{0} \\ \mathbf{0} & \Lambda^i(t) \end{bmatrix} (\mathbf{L}^t(t) \otimes \mathbf{I}_3)\mathbf{x}^i(t). \tag{32}$$

This configuration was applied in the reconfiguration experiment in Section 5.2. Practical application of this strategy to the problem of *separation* in air traffic control is presented in [18, 20].

5. Simulation results

Due to limitation of space, two simulation results are presented for consensus with collision avoidance on the unit sphere, more simulation results are in [18, 20]. The first experiment is to test formation acquisition with avoidance on the sphere. The second experiment is to test arbitrary reconfigurations on the sphere with collision avoidance. Three different communication topologies used are shown in **Figure 4**. In **Figure 4**, Topology 1 (left) is a *fully connected* communication graph with *no* leader, Topology 2 (center) is a *cyclic* communication graph with *one* leader, node 1, and Topology 3 (right) is a *cyclic* communication graph with *no* leader.

Optimization software Sedumi [22] and Yalmip [23] running inside Matlab R2009a, were used for solving all the problems. The simulations were done on an Intel R Core(TM)2 Duo P8600 @ 2.40GHz with 2 GB RAM, running Windows 7.

5.1. Formation acquisition on the unit sphere with avoidance

In this experiment, ten vehicles converge to a formation on the sphere, which is realized by maintaining a relative spacing with each other while also avoiding a static obstacle, with $\alpha = 30^o$. Angle $\beta^{ij} = 20^o$ is set to maintain the relative spacing between the vehicles $\forall\, i, j = 1 \cdots 10, i \neq j$. The initial positions are:

$$\begin{aligned}
x^1(0) &= [0.3417 \quad 0.5555 \quad 0.7581]^T \\
x^2(0) &= [0.4960 \quad 0.1270 \quad 0.8589]^T \\
x^3(0) &= [0.3045 \quad 0.9497 \quad 0.0730]^T \\
x^4(0) &= [0.5735 \quad 0.7952 \quad 0.1967]^T \\
x^5(0) &= [0.8005 \quad 0.3867 \quad 0.4580]^T \\
x^6(0) &= [0.3727 \quad 0.7372 \quad 0.5637]^T \\
x^7(0) &= [0.0355 \quad 0.5117 \quad 0.8585]^T \\
x^8(0) &= [0.6553 \quad 0.7428 \quad 0.1371]^T \\
x^9(0) &= [0.9188 \quad 0.2446 \quad 0.3094]^T \\
x^{10}(0) &= [0.0261 \quad 0.8773 \quad 0.4792]^T
\end{aligned}$$

The result for Topology 1 is shown in **Figure 6** (left), while the right figure shows the result obtained using Topology 3 – a cyclic graph which produces a *circulant* Laplacian **L**, whose dynamics leads to swirling motion. The proof is in [18].

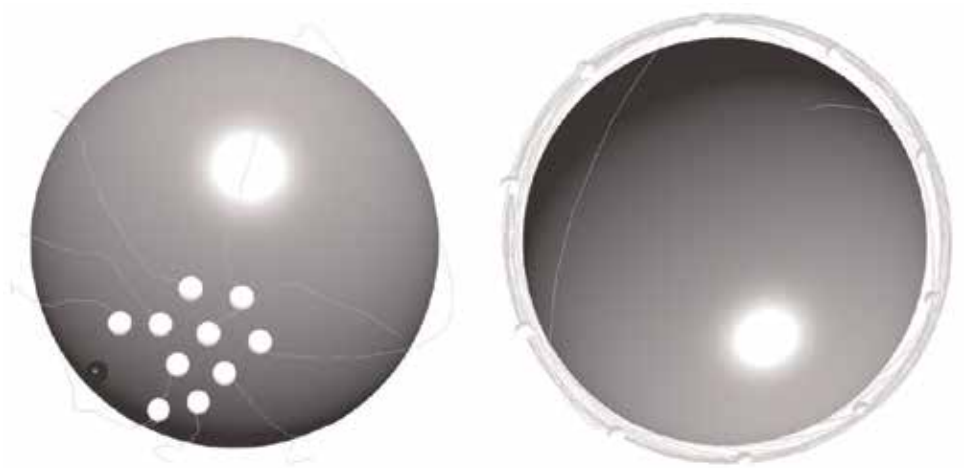

Figure 6. Ten-vehicle formation acquisition using topology 1 (left), and using topology 3 (right).

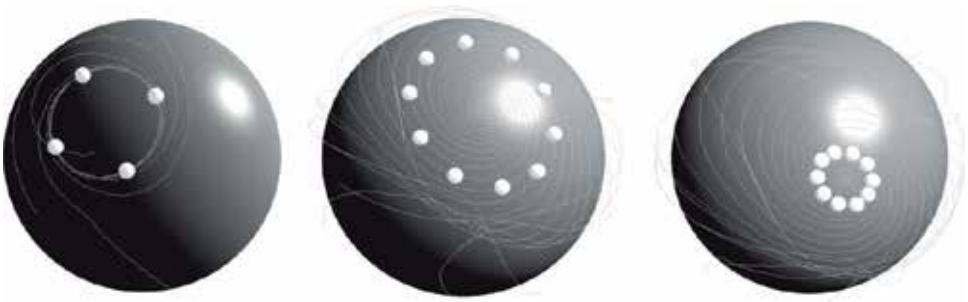

Figure 7. Four-vehicle formation acquisition using topology 3 with $\beta^{ij} = 30^{\circ}$ (left), and ten-vehicle formation acquisition, using topology 3, with $\beta^{ij} = 20^{\circ}$ (center) and $\beta^{ij} = 0^{\circ}$ (right).

When relative spacing are specified between the vehicles, the motion obtained from this Laplacian is like the result obtained in [11]. However, when there is no relative spacing specified, the circular motion converges to a point. Using a circulant matrix such as that of Topology 3, one can vary the radius of the circular formation achieved $\left(r = \cos\theta^{ij}\right)$; by setting θ^{ij} equal for all i, j and varying its size with time. If the magnitude of angle θ is reduced, the radius of the circular formation structure obtained also reduces, and vice versa. **Figure 7** (left) shows the result for setting $\theta^{ij} = 30^{\circ} \forall\, i, j$ for four vehicles. The center and right figures show the results for ten vehicles as θ^{ij} moves gradually from 20° toward 0°. When $\theta^{ij} = 0 \forall\, i, j$, the vehicles rendezvous to a point.

5.2. Collision free reconfiguration on the unit sphere with avoidance of no-fly zones

This is a more traditional reconfiguration problem which we try to solve by using the consensus based protocols presented in this chapter. Three flying vehicles (e.g. UAVs), are required to fly from their initial positions to given final positions. There are *cross-paths* (inter-vehicle

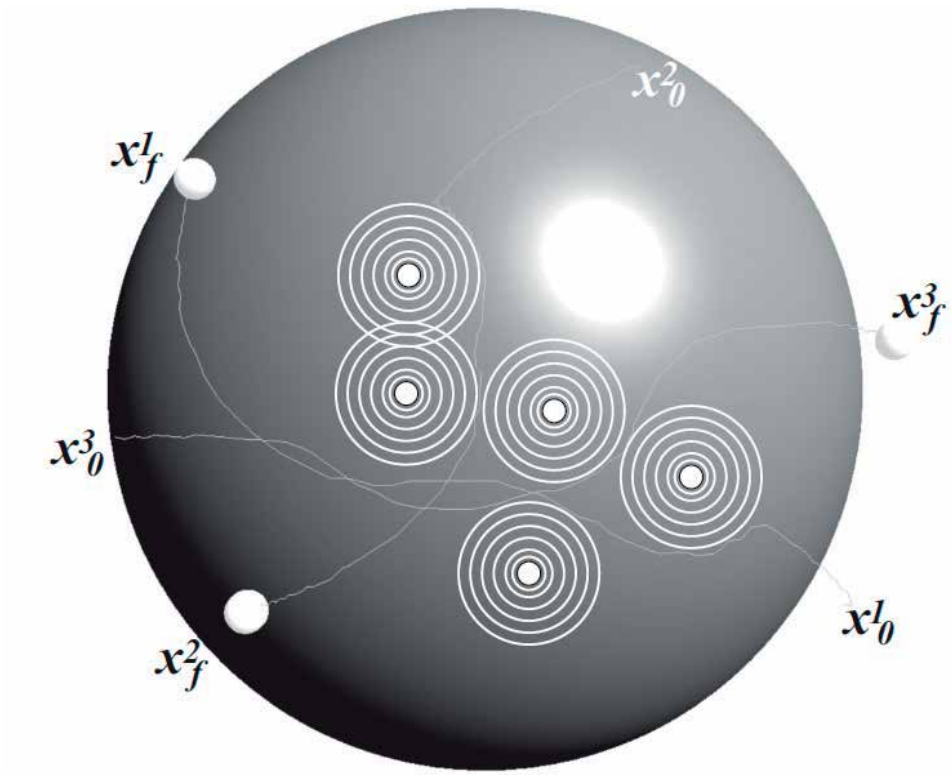

Figure 8. Three-vehicle reconfiguration with collision avoidance and avoidance of no-fly zones.

collision constraints) in addition to *no-fly zones* (static obstacle constraints), between the initial and final positions. The initial positions are:

$$\begin{aligned} x_0^1(0) &= [0.8659 \quad 0 \quad -0.4999]^T \\ x_0^2(0) &= [0.4165 \quad -0.5721 \quad 0.7071]^T \\ x_0^3(0) &= [-0.5878 \quad -0.809 \quad 0]^T \end{aligned}$$

The desired final positions are:

$$\begin{aligned} x_f^1(0) &= [-0.4330 \quad -0.7499 \quad 0.4999]^T \\ x_f^2(0) &= [-0.2939 \quad -0.9045 \quad -0.309]^T \\ x_f^3(0) &= [0.9393 \quad -0.3052 \quad 0.1564]^T \end{aligned}$$

For inter-vehicle collision avoidance, they are required to maintain a minimum safety distance of$r = \cos 10^o$ units. Five no-fly zones are imposed on the vehicles at the following positions:

$$
\begin{aligned}
x_{obs}^{1} &= [0.5237 \quad -0.7208 \quad 0.454]^T \\
x_{obs}^{2} &= [0.2939 \quad -0.9045 \quad -0.309]^T \\
x_{obs}^{3} &= [0 \quad -0.9877 \quad 0.1564]^T \\
x_{obs}^{4} &= [0.5878 \quad -0.809 \quad 0]^T \\
x_{obs}^{5} &= [0 \quad -0.9511 \quad 0.309]^T
\end{aligned}
$$

The radii of the no-fly zones are equal to r, therefore $\beta^{ij} = \alpha^{ij} = 10^{o} \forall\, i, j\ (i \neq j)$ for this simulation. The result is shown in **Figure 8**.

On a final note, we have attempted to solve the problem of consensus in a spherical coordinate system by solving in on the unit sphere. The same unit sphere was used in [11]. This is convenient because the results are easier to visualize and compute on the unit sphere. The results presented here can be applied directly to real-life planetary navigation problems such as *horizontal separation of aircraft* [18, 20], simply by transforming actual position vectors into unit vectors in the unit sphere, solving to obtain the solution trajectories, and transforming the solutions back to actual desired trajectories in the real-world coordinates. The unit of measurement for implementation will therefore depend on the application at hand.

Author details

Innocent Okoloko

Address all correspondence to: okoloko@ieee.org

Department of Electrical Engineering, Universidad de Ingenieria y Tecnologia, Lima, Peru

References

[1] Beard RW, McLain TW, Nelson DB, Kingston D. Decentralized cooperative aerial surveillance using fixed-wing miniature UAVs. Proceedings of the IEEE. 2006;**94**(7):1306-1324. DOI: 10.1109/JPROC.2006.876930

[2] Leonard NE, Paley DA, Lekien F, Sepulchre R, Fratantoni DM, Davis RE. Collective motion, sensor networks and ocean sampling. Proceedings of the IEEE. 2007;**95**(1):48-74. DOI: 10.1109/JPROC.2006.887295

[3] Mesbahi M, Hadaegh H. Formation flying control of multiple spacecraft via graphs, matrix inequalities, and switching. Journal of Guidance, Control, and Dynamics. 2001;**24**(2):369-377. DOI: 10.2514/2.4721

[4] Blackwood G, Lay O, Deininger B, Gudim M, Ahmed A, Duren R, Noeckerb C, Barden B. The StarLight mission: A formation-flying stellar interferometer. In: SPIE 4852, Interferometry in Space; 22 August; Waikoloa, Hawaii. SPIE Digital Library; 2002. DOI: 10.1117/12.460942

[5] Justh EW, Krishnaprasad PS. Equilibria and steering laws for planar formations. Systems and Control Letters. 2004;**52**(1):25-38. DOI: 10.1016/j.sysconle.2003.10.004

[6] Sepulchre R, Pale DA, Leonard NE. Stabilization of planar collective motion: All-to-all communication. IEEE Transactions on Automatic Control. 2007;**52**(5):811-824. DOI: 10.1109/TAC.2007.898077

[7] Chandler PR, Pachter M, Rasmussen S. UAV cooperative control. In: IEEE ACC; 25–27 June; Arlington, VA. IEEEXplore; 2001. pp. 50-55. DOI: 10.1109/ACC.2001.945512

[8] Richards A, Schouwenaars T, How JP, Feron E. Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming. AIAA Journal of Guidance Control and Dynamics. 2002;**25**:755-764. DOI: https://doi.org/10.2514/2.4943

[9] Okoloko I, Basson A. Consensus with collision avoidance: An LMI approach. In: 5th IEEE International Conference on Automation, Robotics and Applications; 06–08 December; Wellington, NZ. IEEE; 2011. ISBN:9781457703287

[10] Okoloko I. Path Planning for Multiple Spacecraft using Consensus with LMI Avoidance Constraints. In: IEEE Aerospace Conference; 3–10 March; Big Sky, Montana. IEEEXplore; 2012. pp. 1-8. DOI: 10.1109/AERO.2012.6187118

[11] Paley D. Stabilization of collective motion on a sphere. Automatica. 2009;**41**:212-216. DOI: 10.1016/j.automatica.2008.06.012

[12] Hernandez S, Paley DA. Stabilization of collective motion in a time-invariant flowfield on a rotating sphere. In: IEEE ACC; St. Louis, MO: IEEEXplore; 2009. pp. 623-628. DOI: 10.1109/ACC.2009.5160631

[13] Justh EW, Krishnaprasad PS. Natural frames and interacting particles in three dimensions. In: Proceedings of Joint 44th IEEE CDC and European control conf.; 12–15 December; Seville, Spain. IEEEXplore; 2005. pp. 2841-2846. DOI: 10.1109/CDC.2005.1582594

[14] Kim Y, Mesbahi M. Quadratically constrained attitude control via semidefinite programming. IEEE Transactions on Automatic Control. 2004;**49**:731-735. DOI: 10.1109/TAC.2004.825959

[15] Biggs N. Algebraic Graph Theory, Cambridge Tracks in Mathematics. 2nd ed. Cambridge University Press; 1974. p. 205. ISBN: 0521458978

[16] Peng L, Zhao Y, Tian B, Zhang J, Bing-Hong W, Hai-Tao Z, Zhou T. Consensus of self-driven agents with avoidance of collisions. Physical Review. 2009;**79**(E). DOI: 10.1103/PhysRevE.79.026113

[17] Olfati-Saber R. Flocking for multi-agent dynamic systems: Algorithms and theory. IEEE Transactions on Automatic Control. 2006;**51**(3):401-420. DOI: 10.1109/TAC.2005.864190

[18] Okoloko I. Multi-path planning and multi-body constrained attitude control [dissertation] PhD Thesis. Stellenbosch, South Africa: Stellenbosch University; 2012. p. 185. Available from: http://hdl.handle.net/10019.1/71905

[19] Ren W, Beard RW, McLain TW. Coordination variables and consensus building in multiple vehicle systems. In: Kumar V, Leonard NE, Morse AS, editors. Lecture Notes in Control and Information Sciences. 309th ed. Berlin: Springer-Verlag; 2005. pp. 171-188 ISSN: 2572-4479

[20] Okoloko I. Consensus Based Distributed Motion Planning on a Sphere. In: IEEE ACC; 17–19 June; Washington DC. IEEEXplore; 2013. pp. 6132-6137. DOI: 10.1109/ACC.2013.6580799

[21] Boyd S, Ghaoui LE, Feron E, Balakrishnan V. Linear Matrix Inequalities in System and Control Theory. Philadelphia, PA: SIAM; 1994. p. 205. ISBN: 0-89871-334-X

[22] Sturm JF. Using SeDuMi 1.02, a Matlab toolbox for optimization over symmetric cones. Optimization Methods and Software. 1998;**11**(12):625-653. DOI: 10.1080/10556789908805766

[23] Yalmip LJ. A toolbox for modelling and optimization in Matlab. In: IEEE CACSD Conference; 2–4 Sept.; Taipei, Taiwan. IEEEXplore; 2004. pp. 284-289. DOI: 10.1109/CACSD.2004.1393890

Chapter 3

Multi-Spacecraft Attitude Path Planning Using Consensus with LMI-Based Exclusion Constraints

Innocent Okoloko

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.71580

Abstract

Space missions involving multi-vehicle teams require the cooperative navigation and attitude slewing of the spacecraft or satellites, for such purposes as interferometry and optimal sensor coverage. This introduces extra constraints of exclusion zones between the spacecraft, in addition to the default exclusion constraints already introduced by damaging or blinding celestial objects. In this work, we present a quaternion-based attitude consensus protocol by using the communication topology of the spacecraft team. By using the Laplacian matrix of their communication graph and a semidefinite program, a synthesis of a time-varying optimal stochastic matrix P is done, which is used to generate various consensus and cooperative attitude trajectories from the initial attitudes of the spacecraft. The concept of quaternion-based quadratically constrained attitude control is then employed to satisfy cone avoidance constraints, where exclusion zones are identified, expressed as linear matrix inequalities (LMI), and solved by semidefinite programming (SDP).

Keywords: attitude path planning, consensus, exclusion, optimization, LMI

1. Introduction

Attitude control is the process of making a spacecraft, e.g. a satellite to point toward a specific direction of interest, and attitude path planning is an essential part of space missions. Some current and future space missions require the deployment of teams of spacecraft for such purposes as interferometry and sensor coverage, e.g. [1, 2]. The general problem of attitude control (AC) is important, not only in the navigation of satellites but also of other spacecraft [3], aircraft, and robots. For this reason, the topic has been studied extensively in the literature, e.g. [4–10].

Attitude path planning is a challenging problem and becomes more challenging when it involves multiple spacecraft. First, they are moving at very high speed in highly dynamic

Notation	Meaning
SCi, SC_i	Spacecraft i
q^i	Attitude quaternion vector of SCi, SC_i, $q^i = [q_1\ q_2\ q_3 \mid\ q_4]^T$
q^{-i} or q^{i*}	Conjugate of q^i
$\overline{q}^i$	Vector part of q^i, $\overline{q}^i = \begin{bmatrix} q_1\ q_2\ q_3 \end{bmatrix}^T$
$\overline{q}^{i\times}$	Antisymmetric of q^i
$\mathbf{q}$	Stacked vector of more than one quaternion vectors
$\mathbf{q}^{off}$	Stacked vector of more than one offset quaternion vectors
Ω, Π	Quaternion dynamics plant matrix
$\mathbf{P}$	Quaternion dynamics Laplacian-like plant matrix
ω	Angular velocity
τ	Control torque
J	Inertia matrix
$\mathbf{L}$	Laplacian matrix
$\mathbf{P}$	Laplacian-like stochastic matrix
$\mathbf{I}_n$	Then $n \times n$ identity matrix
$\mathcal{S}^m$	The set of $m \times m$ positive definite matrices
$\tilde{A}$	Cone avoidance constraint matrix
$\mathcal{R}^i$	Rotation matrix corresponding to q^i
$\mathcal{F}^I_{SC_i}$	Fixed coordinate (Inertial) frame with origin at SC_i's center
$\mathcal{F}^B_{SC_i}$	Rotational coordinate (Body) frame with origin at SC_i's center
$v^B_{obs_i}$	Vector of obstacle in $\mathcal{F}^B_{SC_i}$
$v^I_{obs_i}$	Vector of obstacle in $\mathcal{F}^I_{SC_i}$
$v^I_{obs_i,j}$	Vector of the j^{th} obstacle in $\mathcal{F}^I_{SC_i}$
$v^B_{cam_i}$	Vector of the SC_i's camera in $\mathcal{F}^B_{SC_i}$
$v^I_{cam_i}$	Vector of the SC_i's camera in $\mathcal{F}^I_{SC_i}$
$\otimes$	Kronecker multiplication operator
$\odot$	Quaternion multiplication operator
$\ominus$	Quaternion difference operator
t_0	Initial time
t_f	Final time
x^i	Position vector of SCi, SC_i
$\mathbf{x}$	Stacked vector of n position vectors
$(x^{ij})^{off}$	Offset vector between i and j
$\mathbf{x}^{off}$	Stacked vector of n offset vectors
$\mathcal{C}$	The consensus space for $\mathbf{q}$, $\mathcal{C} = \{\mathbf{q} \mid q^1 = q^2 =, \cdots, = q^n\}$

Table 1. Frequently used notations in this chapter.

environments, subject to external constraints such as blinding celestial objects, which can damage onboard sensors. Secondly, because they are in a team, they must be careful with each other when changing attitude, so as not to collide with each other and damage appendages. We consider a team of networked spacecraft, which share some common objectives, where consensus theory based on graph Laplacians can be applied [11, 12].

Spacecraft attitude dynamics is usually represented by unit quaternions because quaternion dynamics do not encounter the singularities associated with other representations. However, quaternion dynamics are non-linear, which makes it difficult to apply Laplacian-like dynamics directly to quaternions.

Next, we consider some previous work on constrained attitude path planning. In [4], attitude control was formulated as a quadratically constrained optimization problem. Linear matrix inequalities (LMIs) and semidefinite programming (SDP) were employed to solve it for a multiple spacecraft scenario in [6]. In [10], spacecraft attitude stabilization on a sphere was studied. The control torques required for effective attitude stabilization were reduced from three to two. In [12], a consensus-based approach was applied in distributed attitude alignment of a team of communicating spacecraft flying in formation. In [13], a Laplacian-based protocol implemented using the *modified Rodriquez parameters* (MRP) was employed in *leader following* attitude control of spacecraft.

However, none of these aforementioned works apply consensus theory directly to quaternions, except our previous works [7–9]. In addition, only [4, 6–9] tackle the important problem of attitude cone avoidance constraints. Moreover, the works [4, 6, 7] were developed for spacecraft in the same coordinate frame, which does not have a direct practical implementation unless developed further.

To handle the difficulty of nonlinearity in quaternion kinematics, we cast the Q-CAC problem as a semidefinite program, which is subject to convex quadratic constraints, stated as LMI. Then a series of Laplacian-like matrices are synthesized, which satisfy the constraints and enables the spacecraft achieve consensus with exclusion. We employed available optimization software tools such as Sedumi [14] and Yalmip [15] running inside MATLAB®, for simulation.

Moreover, the solution presented here was developed for the realistic scenario of spacecraft in different coordinate frames, making it practical to implement directly. Therefore, the contributions of this chapter are aspects of our previous works [7–9], which are: (1) development of a quaternion consensus protocol; (2) incorporating dynamic cone avoidance constraints into the consensus framework; (3) providing a mathematical convergence analysis for the quaternion-based consensus framework; (4) extending the approach to multiple spacecraft in any coordinate frames, thereby making it more suitable for practical implementation.

The rest of the chapter is organized as follows: The problem statement is in Section 2, followed by brief mathematical preliminaries in Section 3. The solution technique and convergence analysis are in Section 4, numerical simulations in Section 5, and conclusion in Section 6. Notations frequently used in this chapter are listed in **Table 1**. The words *obstacle, avoidance, exclusion, exclusion vector* may be used interchangeably in this chapter.

2. Problem statement

The problem of attitude reconfiguration of a team of communicating spacecraft with avoidance constraints can be stated as follows. Given a set of communicating spacecraft with initial positions at $x^i(t_0) \in \mathbb{R}^3$ $i=1\cdots n$, initial attitudes represented by quaternions $q^i(t_0)$, generate a sequence of consensus trajectories that drive the team to a consensus attitude $q(t_f)$ while satisfying exclusion, avoidance and norm constraints.

There are two aspects of the problem stated above: the first is a *consensus* problem, wherein it is desired to drive the attitudes to a collective consensus attitude, or to various formation attitudes. For bare consensus, the final consensus is that each spacecraft should eventually point to the average of the initial attitudes. However, *relative offset quaternions* can be applied so the consensus attitude can be a desired formation attitude, e.g. each spacecraft can point at 5° away from each other about the z axis. The second problem is that of *avoidance* constraints. This is also important for the team, because spacecraft usually have appendages, some have thrusters that emit plumes, and some have instruments that can be damaged by blinding celestial objects or by the appendage or plume of a team member.

However, the ordinary consensus protocol was not developed for quaternion dynamics. It violates the non-linearity of quaternion kinematics and the quaternion norm preserving requirement. Moreover, the ordinary consensus algorithm also does not incorporate collision avoidance in *adversarial* situations; this is a Q-CAC problem. Thus, in this paper, we present aspects of our previous works [7–9, 16], where we combined consensus theory with constrained optimization to solve the problems stated above. We cast the problems as a *semidefinite program* (SDP), which is augmented with some convex quadratic constraints written as *linear matrix inequalities* (LMI).

We present a quaternion consensus protocol that computes a consensus attitude trajectory each time step, and a Q-CAC optimization procedure, which decides whether it is safe to follow the computed attitude trajectory or not. When generated trajectories are unsafe, it

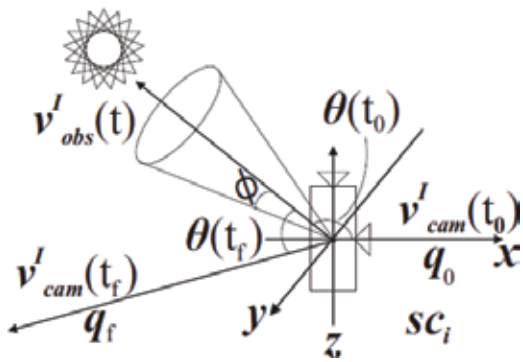

Figure 1. Constrained attitude control problem for a single-spacecraft single-exclusion scenario.

computes a new set of quaternion vectors that avoid collision and the cycle repeats until consensus is achieved.

To understand the avoidance aspect, we begin with a simpler illustration of the spacecraft Q-CAC problem with a single spacecraft and a single obstacle (exclusion) vector, as shown in **Figure 1**.

Let SC_i denote spacecraft i, and $v^I_{cam_i}(t)$ denote the unit camera vector in $\mathcal{F}^I_{SC_i}$ corresponding to the SC_i's attitude q^i (see **Table 1** for definitions). Also, let $v^I_{obs_i}(t)$ be the attitude quaternion representing the obstacle to be avoided (e.g. the Sun, as shown in **Figure 1**). It is desired that the time evolution of camera vector $v^I_{cam_i}(t_0)$ to $v^I_{cam_i}(t_f)$ should avoid $v^I_{obs_i}(t)$ always, while maintaining a minimum angular separation of $\varnothing$. The requirement can therefore be stated as

$$\theta(t) \geq \varnothing \tag{1}$$

or

$$\begin{gathered} v^I_{cam_i}(t)^T v^I_{obs_i}(t) \leq \cos\varnothing, \\ \forall t \in [t_0, t_f] \end{gathered} \tag{2}$$

The constraint is *non-convex* and *quadratic* and should be *convexified* for it to be represented as a LMI. The convexification was provided in [4], using the quaternion attitude constraints formulation developed in [3] for a single-spacecraft single-obstacle scenario. For that solution, v^I_{obs} was static, $v^I_{cam_i}(t)$ was evolving, and both vectors were in the same coordinate frame. This makes it incomplete for practical implementation because, in reality the obstacle and spacecraft are in different coordinate frames.

In [7–9, 16], we extended the previous avoidance solution to multiple spacecraft. Then we developed a consensus theory of quaternions and appended the new avoidance protocols. We further solved the problem for spacecraft and dynamic obstacles in different coordinate frames to make the solutions more suitable for practical implementation. Next, we present the basic mathematical preliminaries.

3. Mathematical background

In this section, we consider the two basic mathematical theories relevant to this chapter.

3.1. Quaternion-based rotational dynamics

It is convenient to use unit quaternions to represent the attitude of a rigid body rotating in three-dimensional space (such as spacecraft or satellite) because quaternions are not susceptible to the problems of singularities inherent in using Euler angles [17].

The quaternion is a four-element vector

$$q = \left[q_1\ q_2\ q_3 | q_4\right]^T. \tag{3}$$

Here, $[q_1\ q_2\ q_3]^T$ is a vector representing the axis of rotation in the Cartesian (x, y, z) coordinates and q_4 is a scalar representing the angle of rotation, of the quaternion. The difference between two quaternions q^1 and q^2 can be represented in multiplication terms as

$$\begin{aligned} q^d &= q^1 \odot q^{-2} = q^1 \odot \left[-q_1^2\ \ -q_2^2\ \ -q_3^2\ \ -q_4^2\right]^T \\ &= Q^2 q^1, \end{aligned} \tag{4}$$

where q^{-2} is the conjugate of q^2 and $\odot$ is defined in **Table 1**. Q^2 is defined as

$$Q^i = \begin{bmatrix} q_4^i & q_3^i & -q_2^i & -q_1^i \\ -q_3^i & q_4^i & q_1^i & -q_2^i \\ q_2^i & -q_1^i & q_4^i & -q_3^i \\ q_1^i & q_2^i & q_3^i & q_4^i \end{bmatrix} \tag{5}$$

It follows that the transformation of q^1 to q^2 was achieved by the rotation quaternion q^d.

The rotational dynamics for the i^{th} quaternion is

$$\dot{q}^i = \frac{1}{2}\Omega^i q^i = \frac{1}{2}\Pi^i \omega^i, \tag{6}$$

where

$$\Omega^i = \begin{bmatrix} 0 & \omega_3^i & -\omega_2^i & \omega_1^i \\ -\omega_3^i & 0 & \omega_1^i & \omega_2^i \\ \omega_2^i & -\omega_1^i & 0 & \omega_3^i \\ -\omega_1^i & -\omega_2^i & -\omega_3^i & 0 \end{bmatrix} \tag{7}$$

$$\Pi^i = \begin{bmatrix} -q_4^i & q_3^i & -q_2^i \\ -q_3^i & -q_4^i & q_1^i \\ q_2^i & -q_1^i & -q_4^i \\ q_1^i & q_2^i & q_3^i \end{bmatrix} \tag{8}$$

Euler's first-order discretization of Eq. (6) gives

$$q^i(k+1) = q^i(k) + \frac{\Delta t}{2}\Pi^i(k)\omega^i(k). \tag{9}$$

The dynamics of the rotational velocity ω^i is

$$\begin{bmatrix}\dot{\omega}_1^i\\ \dot{\omega}_2^i\\ \dot{\omega}_3^i\end{bmatrix} = \begin{bmatrix}\left(\left(J_2^i - J_3^i\right)\omega_2^i\omega_3^i + \tau_1^i\right)/J_1^i\\ \left(\left(J_3^i - J_1^i\right)\omega_3^i\omega_1^i + \tau_2^i\right)/J_2^i\\ \left(\left(J_1^i - J_2^i\right)\omega_1^i\omega_2^i + \tau_3^i\right)/J_3^i\end{bmatrix}$$
$$= \underbrace{\begin{bmatrix}0 & \frac{J_2^i}{J_1^i}\omega_3^i & -\frac{J_3^i}{J_1^i}\omega_2^i\\ \frac{J_3^i}{J_2^i}\omega_3^i & 0 & -\frac{J_1^i}{J_2^i}\omega_1^i\\ \frac{J_1^i}{J_3^i}\omega_2^i & -\frac{J_2^i}{J_3^i}\omega_1^i & 0\end{bmatrix}}_{\Upsilon^i}\begin{bmatrix}\omega_1^i\\ \omega_2^i\\ \omega_3^i\end{bmatrix} + \underbrace{\begin{bmatrix}1/J_1^i & 0 & 0\\ 0 & 1/J_2^i & 0\\ 0 & 0 & 1/J_3^i\end{bmatrix}}_{J^i}\begin{bmatrix}\tau_1^i\\ \tau_2^i\\ \tau_3^i\end{bmatrix} \tag{10}$$

Euler's first-order discretization of Eq. (10) is

$$\omega^i(k+1) = \left(\mathbf{I}_3 + \Delta t\Upsilon^i(k)\right)\omega^i(k) + \Delta t J\tau^i(k), \tag{11}$$

where J_j^i is the moment of inertia, ω_j^i is the rotational velocity and τ_j^i is the control torque, along the three principal axes $j=1,2,3$, for the i^{th} rigid body. Combining Eqs. (9) and (11) in stacked vector form yields

$$\underbrace{\begin{bmatrix}\tau^i(k)\\ \omega^i(k+1)\\ q^i(k+1)\end{bmatrix}}_{\mathfrak{T}(k+1)} = \underbrace{\begin{bmatrix}\Xi^i(k) & \Psi^i(k)\\ \mathbf{I}_3 + \Delta t\Upsilon^i(k) + \Delta t J\Xi^i(k) & \Delta t J\Psi^i(k)\\ \frac{\Delta t}{2}\Pi^i(k) & \mathbf{I}_4\end{bmatrix}}_{F(k)}\underbrace{\begin{bmatrix}\omega^i(k)\\ q^i(k)\end{bmatrix}}_{\mathfrak{T}(k)} \tag{12}$$

It is the task of controller synthesis to determine the Ξ^i and Ψ^i to obtain the torque τ^i that stabilizes the system.

3.2. Basic consensus theory

The *Consensus*-based algorithms are distributed protocols based on *communication graphs*, which can drive the states of a team of communicating agents to an agreed state or a common state. The agents (or vehicles) i ($i=1,\cdots,n$) are represented by vertices of the graph and the edges of the graph are the communication links between them. Denote the state of a vehicle i as x^i; $\mathbf{x}$ is the stacked vector of the states all vehicles in the team, then for systems modeled by first-order dynamics, the following first-order consensus protocol (or similar protocols) have been proposed, e.g. [18, 19]

$$\dot{\mathbf{x}}(t) = -\mathbf{L}\left(\mathbf{x}(t) - \mathbf{x}^{off}\right). \tag{13}$$

When $\|x^i - x^j\| \to (x^{ij})^{off}$ as $t \to \infty$, $\forall i \neq j$ then consensus has been achieved. A more comprehensive presentation of the mathematical basis of consensus (including graph theory), can be found in [16].

However, the basic consensus protocol Eq. (13) cannot admit quaternions directly. Thus, to extend Eq. (13) to attitude quaternions, the following consensus protocol for quaternions was proposed in [7]

$$\dot{\mathbf{q}}(t) = -\mathbf{P}(t)\left(\mathbf{q}(t) \ominus \mathbf{q}^{-off}\right), \tag{14}$$

where $\mathbf{P}(t)$ is a Laplacian-like matrix and $\mathbf{q}(t) = [q^1(t), q^2(t) \cdots q^n(t)]^T$. More analysis of $\mathbf{P}(t)$ follows in the next sections.

4. Solutions

We present solutions to the problem statement in Section 2 [7–9, 16]. The solution involves four steps: (1) synthesis of consensus attitudes for multiple spacecraft; (2) formulation of Q-CAC in different coordinate frames; (3) determining obstacle vectors in different coordinate frames; (4) integration for consensus based Q-CAC.

4.1. Synthesis of consensus attitudes for multiple spacecraft

To develop consensus for quaternions, we adopt an optimization approach. The Laplacian-like stochastic matrix $\mathbf{P}(t)$ in Eq. (14) is synthesized (by an optimization process) at each time step to drive $\mathbf{q}(t)$ to consensus while satisfying quaternion kinematics. The components of $\mathbf{P}(t)$ are

$$\mathbf{P}(t) = \underbrace{\begin{bmatrix} \Lambda^1(t) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \Lambda^n(t) \end{bmatrix}}_{\Lambda(t)} \underbrace{\begin{bmatrix} l_{11}\mathbf{I}_4 & \cdots & l_{1n}\mathbf{I}_4 \\ \vdots & \ddots & \vdots \\ l_{n1}\mathbf{I}_4 & \cdots & l_{nn}\mathbf{I}_4 \end{bmatrix}}_{\Gamma = \mathbf{L} \otimes \mathbf{I}_4}, \tag{15}$$

where $\Lambda^i(t) > 0$ is an unknown positive definite optimization matrix variable, whose components are chosen by the optimization process. Matrix Γ is composed of components of the Laplacian $\mathbf{L} = [l_{ij}]$ $(i, j = 1, \cdots, n)$, which gives $\mathbf{P}(t)$ its Laplacian-like behavior.

We shall now prove the stability of $\mathbf{P}(t)$, i.e. that Eq. (14) does indeed achieve consensus. Note that all the theorems, lemmas and proofs in this section had been presented in [7].

Recall the following standard result on a *matrix pencil* [20].

Theorem 1: For a *symmetric-definite* pencil $\mathbf{A} - \lambda\mathbf{B}$, there exists a nonsingular $S = [s_1, \cdots, s_n]$ such that

$$S^T\mathbf{A}S = \mathrm{diag}(a_1, \cdots, a_n) = D_A, \tag{16}$$

$$S^T\mathbf{B}S = \mathrm{diag}(b_1, \cdots, b_n) = D_B. \tag{17}$$

Moreover, $\mathbf{A}s_i = \lambda_i\mathbf{B}s_i$ for $i = 1, \cdots, n$, where $\lambda_i = a_i/b_i$.

Lemma 1: For any time t, the eigenvalues of $\mathbf{P}(t)$ are $\gamma_i\eta_i(t)$. Here, γ_i are the eigenvalues of Γ and $\eta_i(t)$ the eigenvalues of $\Lambda(t)$. It can therefore be observed that $\mathbf{P}(t)$ has only four zero eigenvalues, the rest of its eigenvalues are strictly positive.

Proof: To find the eigenvalues of $\mathbf{P}(t)$, consider a scalar λ such that for some nonzero vector s

$$\Gamma s = \lambda\Lambda^{-1}(t)s. \tag{18}$$

Eq. (18) defines a *symmetric-definite generalized eigenvalue problem* (SDGEP), where $\Gamma - \lambda\Lambda^{-1}(t)$ defines a matrix pencil. Theorem 1 therefore immediately implies that the eigenvalues of $\mathbf{P}(t)$ are $\gamma_i\eta_i(t)$. One can also easily observe that due to the property of the Laplacian matrix $\mathbf{L}$, $\mathbf{P}(t)$ has positive eigenvalues except for four eigenvalues. This proves the claim.

Theorem 2: The time-varying system Eq. (14) achieves consensus.

Proof: for simplicity, let us assume that there are no offsets, i.e. $\mathbf{q}^{off} = 0$ (or $(q^{off})^i = [0\ 0\ 0\ 1]^T \forall\, i$). Note, when $\mathbf{q}$ has entered the *consensus space* $\mathcal{C} = \left\{\mathbf{q} | q^1 = q^2 =, \cdots, = q^n\right\}$, then $\dot{\mathbf{q}} = 0$. $\mathcal{C}$ is the *nullspace* of $\mathbf{P}(t)$, i.e. the set of all $\mathbf{q}$ such that $\mathbf{P}(t)\mathbf{q} = 0$. Therefore, once $\mathbf{q}$ enters $\mathcal{C}$ it stays there.

Suppose that $\mathbf{q}$ has not entered $\mathcal{C}$, then consider a Lyapunov candidate function $V = \mathbf{q}^T\Gamma\mathbf{q}$; $V > 0$ unless $\mathbf{q} \in \mathcal{C}$. Then,

$$\begin{aligned}
\dot{V} &= \mathbf{q}^T\Gamma\dot{\mathbf{q}} + \dot{\mathbf{q}}^T\Gamma\mathbf{q}, \\
&= -\mathbf{q}^T\Gamma\mathbf{P}(t)\mathbf{q} - \mathbf{q}^T\mathbf{P}(t)^T\Gamma\mathbf{q}, \\
&= -\mathbf{q}^T\Gamma\Lambda(t)\Gamma\mathbf{q} - \mathbf{q}^T\Gamma\Lambda(t)\Gamma\mathbf{q}, \\
&= -2\mathbf{q}^T\Gamma\Lambda(t)\Gamma\mathbf{q}, \\
&= -2z^T\Lambda(t)z,
\end{aligned} \tag{19}$$

where $z = \Gamma\mathbf{q} \neq 0$ for $\mathbf{q} \notin \mathcal{C}$. This implies that $\mathbf{q}$ approaches a point in $\mathcal{C}$ as $t \to \infty$, which proves the claim. Eq. (19) is true as long as $\mathbf{L}$ is nonempty, i.e. if some vehicles can sense, see, or communicate with each other all the time.

4.2. Formulation of Q-CAC in different coordinate frames

Any rigid appendage attached to the body of the i^{th} spacecraft, e.g. a camera, whose pointing direction is $v^I_{cam_i}$ in inertial frame, can be transformed to the spacecraft fixed body frame by the rotation

$$v^B_{cam_i}(t) = \mathcal{R}_i^{-1}(t)v^I_{cam_i}(t), \tag{20}$$

where

$$\mathcal{R}_i(t) = \left(\left(2q_4^i(t)\right)^2 - 1\right)\mathbf{I}_3 + 2\overline{q}^i(t)\overline{q}^i(t)^T - 2q_4^i(t)\overline{q}^i(t)^\times \tag{21}$$

is the rotation matrix corresponding to the $q^i(t)$ at time t; $\overline{q}^i(t)^\times$ is the *antisymmetric matrix* [21].

For simplicity let us consider a single SC_i with a single camera, $v^I_{cam_i}$, and m (possibly, time-varying) obstacles, $v^I_{obs_i.j}$ $(j = 1, \cdots, m)$, defined in $\mathcal{F}^I_{SC_i}$, to be avoided by $v^I_{cam_i}$ when SC_i is re-orientating. Then according to [3], the resulting attitude constraint of Eq. (2) can be written as

$$q^i(t)^T \widetilde{A}^i_j(t) q^i(t) \leq 0. \tag{22}$$

Its LMI equivalent according to [4] is

$$\begin{bmatrix} \mu & q^i(t)^T \\ q^i(t) & \left(\mu \mathbf{I}_4 + \tilde{A}^i_j(t)\right)^{-1} \end{bmatrix} \geq 0. \tag{23}$$

In Eq. (23), μ is chosen to ensure $\mu \mathbf{I}_4 + \tilde{A}_j i(t)$ is positive definite, and

$$\widetilde{A}^i_j(t) = \begin{bmatrix} A_j(t) & b_j(t) \\ b_j(t)^T & d_j(t) \end{bmatrix} \in \mathbb{R}^{4\times 4}, \tag{24}$$

where

$$A_j(t) = v^B_{cam_i}(t) v^I_{obs_i.j}(t)^T + v^I_{obs_i.j}(t) v^B_{cam_i}(t)^T - \left(v^B_{cam_i}(t)^T v^I_{obs_i.j}(t) + \cos\theta\right)\mathbf{I}_3, \tag{25}$$

$$b_j(t) = -v^B_{cam_i}(t) \times v^I_{obs_i.j}(t), \tag{26}$$

$$d_j(t) = v^B_{cam_i}(t)^T v^I_{obs_i.j}(t), \tag{27}$$

for $j = 1, \cdots, m$.

Eq. (22) defines the set of attitude quaternions $q^i(t)$ to satisfy the constraint $v^I_{cam_i}(t)^T v^I_{obs_i.j}(t) \geq \varnothing$ $\forall t \in [t_0, t_f]$, so it is used to find a collision-free $v^I_{cam_i}(t)$. However, in practical situations, another SC (SC_j) can be near SC_i, then another obstacle vector $v^I_{obs_j}(t)$ (e.g. a thruster vector emanating from SC_j) defined in $\mathcal{F}^I_{SC_j}$ should be avoided by SC_i. To address such a practical issue, we present a mechanism to calculate $v^I_{obs_i.j}$ (defined in $\mathcal{F}^I_{SC_i}$) corresponding to $v^I_{obs_j}$ (defined in $\mathcal{F}^I_{SC_j}$) ($v^I_{obs_i.j}$ means the obstacle vector originated from the rotating frame of SC_j but defined in $\mathcal{F}^I_{SC_i}$). Essentially, the mechanism determines the intersection point of $v^I_{obs_j}(t)$ with the sphere of radius r, centered on SC_i. If such an intersection exists, it defines $v^I_{obs_i.j}$ which can be used to define an attitude constraint represented as Eq. (22) to be avoided by SC_i.

Figure 2 illustrates the scenario. SC_1 and SC_2 are in their different coordinate frames relative to Earth. A thruster attached to SC_1 body frame is at $v^I_{obs_1}$, while the circles around SC_1 and SC_2 are spheres representing the coordinate frames from which their attitude evolves. If both spacecraft are close enough, then vector $v^I_{obs_1}$ may intersect a point on the sphere of SC_2,

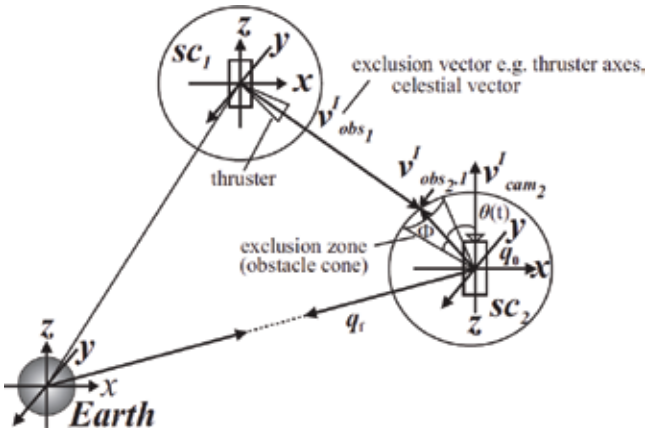

Figure 2. Q-CAC problem in different frames.

whereby the intersection defines $v^I_{obs_2.1}$ in the frame of SC_2. The requirement is that as SC_2 changes its attitude from q_0 to q_f, $v^I_{cam_2}$ must avoid the cone created around $v^I_{obs_2.1} \forall t \in [t_0, t_f]$.

4.3. Determination of obstacle vectors in different coordinate frames

Given SC_i in $\mathcal{F}^I_{SC_i}$ and SC_j in $\mathcal{F}^I_{SC_j}$, with emanating vectors, one can easily determine an intersection between a vector emanating from $\mathcal{F}^I_{SC_j}$ with the sphere centered on $\mathcal{F}^I_{SC_i}$ by using onboard sensors, or by application of computational geometry. Given a line segment originating at p_1 and terminating at p_2, a point $p = [p_x\ p_y\ p_z]^T$ on $[p_1, p_2]$ can be tested for intersection with a sphere centered at p_3 with radius r [22]. Thus, for any $v^I_{obs_j}(t)$ in $\mathcal{F}^I_{SC_j}$, if an intersection point $p(t)$ exists at time t with the sphere centered on $\mathcal{F}^I_{SC_i}$ with radius r, then $v^I_{obs_i.j}(t) = p(t)$; otherwise, one can set $v^I_{obs_i.j}(t) = -v^I_{cam_i}(t)$, to show that no constraints violation has occurred. The value of r will therefore depend on the application at hand, but must be proportional to the urgency of avoiding obstacle vectors originating from other spacecraft.

The above formulation effectively decentralizes the problem. Therefore, each spacecraft can solve the problem by communicating with its neighbors and/or using its own sensors. Euler first order discretization of Eq. (14) is

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \Delta t \dot{\mathbf{q}}_k = \mathbf{q}_k - \Delta t \mathbf{P}(t)\mathbf{q}_k. \tag{28}$$

The decentralized dynamics for any SC_i is therefore

$$q^i_{k+1} = q^i_k - \Delta t \underbrace{\left[y\Lambda^i_1(t) - \Lambda^i_2(t) \cdots - \Lambda^i_y(t) \right]}_{\mathbf{P}^i(t)} \left[q^T_1(t)\ \ q^T_2(t) \cdots q^T_y(t) \right]^T, \tag{29}$$

where $q_1^T(t)\ q_2^T(t)\cdots q_y^T(t)$ are the quaternions of the y other neighboring SC, which SC_i can communicate with at time t. Moreover, since we are going to apply consensus quaternion protocol Eq. (29), norm constraints must be enforced as follows:

$$q_k^i T\left(q_{k+1}^i - q_k^i\right) = 0 \tag{30}$$

Eq. (30) is the discrete time version of $q^i(t)^T \dot{q}^i(t) = 0$ or $\mathbf{q}(t)^T \dot{\mathbf{q}}(t) = 0$. This guarantees that $q^i(t)^T q^i(t) = 1$ or $\mathbf{q}(t)^T \mathbf{q}(t) = n$ for SC, iff $\|q^i(0)\| = 1\ \forall\, i$.

4.4. Integration for consensus based Q-CAC

Using semidefinite programming, the solutions presented previously be cast as an optimization problem, augmented with a set of LMI constraints, and solved for optimal consensus quaternion trajectories. We consider the algorithm in discrete time. Given the initial attitude $q^i(0)$ of SC_i, $(i = 1, \cdots, n)$, find a sequence of consensus quaternion trajectories that satisfies the following constraints:

$$q_{k+1}^i = q_k^i - \Delta t \mathbf{P}^i(t) q_k^i, \qquad \text{quaternion consensus dynamics constraint} \tag{31}$$

$$q_k^i T\left(q_{k+1}^i - q_k^i\right) = 0, \qquad \text{norm constraint} \tag{32}$$

$$\begin{bmatrix} \mu & q^i(t)^T \\ q^i(t) & \left(\mu \mathbf{I}_4 + \tilde{A}_j^i(t)\right)^{-1} \end{bmatrix} \geq 0 \quad \text{exclusion constraints} \tag{33}$$

Once the next safe quaternion trajectory q_{safe}^i has been determined, the control torque τ^i and angular velocity ω^i to rotate the SC_i optimally to q_{safe}^i can be determined by using the normal quaternion dynamics Eq. (12).

5. Simulation results

Due to limitation of space, we present three results for attitude multi-path planning in different coordinate frames. More results can be found in [7, 8, 16].

5.1. Dynamic avoidance in different coordinate frames without consensus

In this experiment, SC_1 and SC_2 are attempting a reconfiguration to Earth (either changing orientation to Earth or pointing an instrument to Earth). The initial quaternions of SC_1 and SC_2 are $q_0^1 = q_0^2 = [0\ \ 0\ \ 0\ \ 1]^T$. The desired final quaternions are

$$\begin{aligned} q_f^1 &= [0.2269\ \ 0.0421\ \ 0.9567\ \ 0.1776]^T \\ q_f^2 &= [0\ \ 0\ \ 0.9903\ \ 0.1387]^T \end{aligned} \tag{34}$$

Three thrusters of SC_1 in $\mathcal{F}_{SC_1}^B$ are

$$
\begin{aligned}
v^B_{obs_1.1} &= [-0.2132 - 0.0181 \;\; 0.9768]^T \\
v^B_{obs_1.2} &= [0.314 \;\; 0.283 - 0.906]^T \\
v^B_{obs_1.3} &= [-0.112 - 0.133 - 0.985]^T
\end{aligned} \tag{35}
$$

A single thruster of SC_2 in $\mathcal{F}^B_{SC_2}$ is at

$$
v^B_{obs_2} = [0.02981 \;\; 0.0819 \;\; 0.9962]^T \tag{36}
$$

We want $v^I_{obs_2}$ to avoid $v^I_{obs_1.1}$ by 50°, and avoid $v^I_{obs_1.2}$ and $v^I_{obs_1.3}$ by 30° while both are maneuvering to their desired final attitudes. **Figure 3(a)** shows the avoidance between thrusters of SC_1 and SC_2 during reorientation to Earth: SC_2 cannot reconfigure to the desired q^2_f due to the avoidance constraints. Note that $v^I_{obs_2.1}$, $v^I_{obs_2.2}$, $v^I_{obs_2.3}$ are the points of intersections of $v^I_{obs_1.1}$, $v^I_{obs_1.2}$, $v^I_{obs_1.3}$ with SC_2. **Figure 3(b)** satisfaction of avoidance constraints: the sudden jumps to and from -1 indicate times when any of $v^I_{obs_1.1}$, $v^I_{obs_1.2}$, $v^I_{obs_1.3}$ lost intersection with the sphere of SC_2 and therefore was replaced with $-v^I_{obs_1.i}$, $i = 1, \cdots, 3$.

This experiment demonstrates that when both constraints are in conflict the avoidance constraint is superior to the desired final quaternion constraint. As seen from (a), SC_2 cannot reconfigure exactly to the desired q^2_f due to the satisfaction of the avoidance constraints. To resolve this, it is necessary to change either the position of SC_2 or SC_1.

5.2. Consensus-based dynamic avoidance in different coordinate frames

In this experiment, SC_1, SC_2, and SC_3 will maneuver to a consensus attitude. The initial positions are

$$
\begin{aligned}
\mathcal{F}^I_{SC_1} &= [-2 \;\; 0 \;\; 2]^T \\
\mathcal{F}^I_{SC_2} &= [0.5 \;\; 0 \;\; 2]^T \\
\mathcal{F}^I_{SC_3} &= [3 \;\; 0 \;\; 2]^T
\end{aligned} \tag{37}
$$

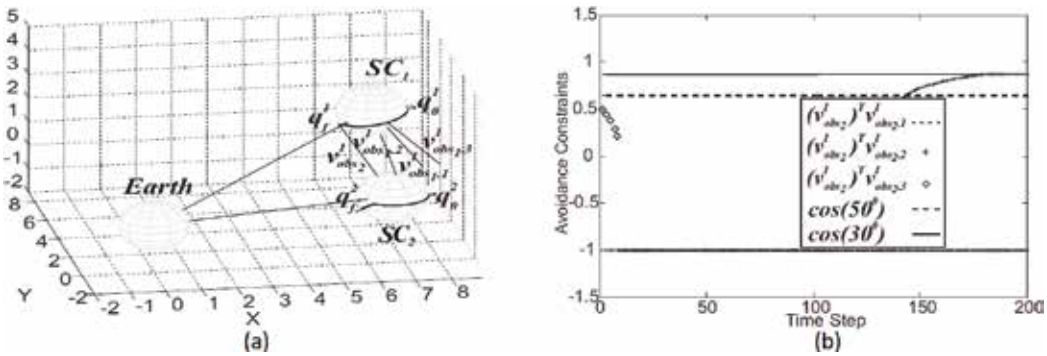

Figure 3. Reconfiguration of two spacecraft with avoidance in different coordinate frames: (a) the trajectories, (b) the avoidance graph.

A set of initial quaternions were randomly generated, with the following data:

$$\begin{aligned} q_0^I &= [-0.5101 \;\; 0.6112 - 0.3187 - 0.5145]^T \\ q_0^2 &= [-0.9369 \;\; 0.2704 - 0.1836 - 0.124]^T \\ q_0^3 &= [0.1448 - 0.1151 \;\; 0.1203 \;\; 0.9753]^T \end{aligned} \tag{38}$$

In the direction of the initial attitude q_0^i of each SC_i, a sensitive instrument $v_{cam_i}^I$ is attached. Also, each SC_i has a thruster pointing to the opposite of q_0^i. It is desired that $v_{cam_i}^I$ avoids the thruster plumes emanating from each of the two other SC by 30° during the entire period of the maneuvers. From the generated initial quaternions, there is possibility of intersection of the thrusters of SC_1 and SC_3, with SC_2, and the thruster of SC_2 may damage SC_1 or SC_3 at any time k. **Figure 4(a)** shows the solution trajectories; (b) shows the avoidance graph, which shows that constraints are not violated; (c) shows the consensus graph. The final consensus quaternion is $q_f = [-0.8167 \;\; 0.4807 - 0.2396 \;\; 0.2112]^T$. This is the normalized average of the initial attitude quaternions, which proves that consensus is achieved.

5.3. Consensus-based attitude formation acquisition with avoidance

To test the capability of the consensus algorithm in formation acquisition, SC_1, SC_2, and SC_3 will maneuver to a consensus formation attitude. Relative offset quaternions were defined to enable the sensitive instruments to point at 30° offsets from each other about the z-axis. The previous set of initial data for q_0^i and $\mathcal{F}_{SC_i}^I$ were used. The relative offsets are

$$\begin{aligned} q_1^{off} &= [0 \;\; 0 \;\; 0 \;\; 1]^T \\ q_2^{off} &= [0 \;\; 0 \;\; 0.2588 \;\; 0.9659]^T \\ q_3^{off} &= [0 \;\; 0 \;\; 0.5 \;\; 0.866]^T \end{aligned} \tag{39}$$

Like the previous experiment, we want the sensitive instruments to avoid the thruster plumes emanating from each of the two other SC by an angle of 30°. The trajectories are shown in **Figure 5(a)** and **(b)** shows the avoidance graph; no constraints are violated, and (c) shows the consensus graph. The final consensus quaternions are

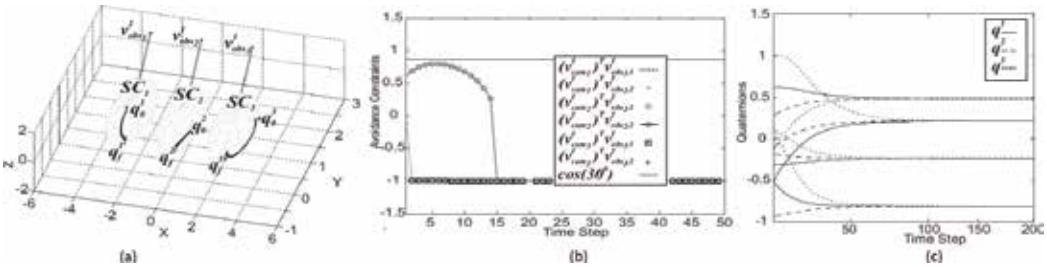

Figure 4. Consensus-based dynamic avoidance in different coordinate frames. (a) Reorientation to consensus attitude with intervehicle thruster plume avoidance, (b) avoidance constraints graph, (c) attitude consensus graph.

$$
\begin{aligned}
q_f^1 &= [-0.6926 \;\; 0.6468 - 0.2798 \;\; 0.1541]^T \\
q_f^2 &= [-0.8364 \;\; 0.4455 - 0.2303 \;\; 0.2212]^T \\
q_f^3 &= [-0.9232 \;\; 0.2138 - 0.1652 \;\; 0.2733]^T
\end{aligned} \tag{40}
$$

The differences of these quaternions are 30° apart about the same axis.

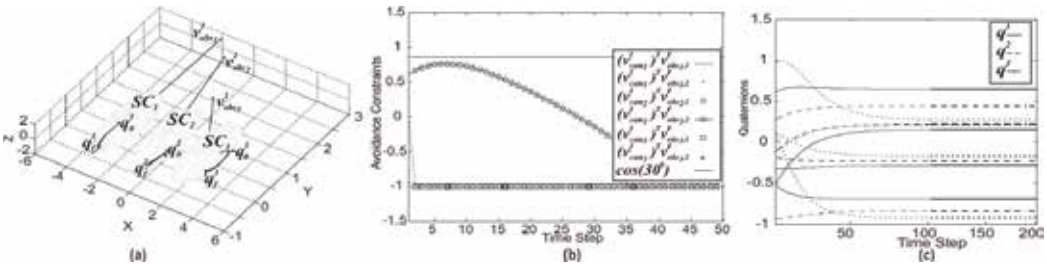

Figure 5. Consensus-based attitude formation acquisition with avoidance. (a) Reorientation to consensus formation attitude with intervehicle thruster plume avoidance, (b) avoidance constraints graph, (c) attitude consensus graph.

6. Conclusion

We presented a solution, which we previously developed, to the problem of attitude path planning for multiple spacecraft with avoidance of exclusion zones, by combining consensus theory and Q-CAC optimization theory. Using the solutions, a team of spacecraft can point to the same direction, or to various formation patterns, while they avoid an arbitrary number of attitude obstacles or exclusion zones in any coordinate frames. We also provided the proof of stability of the Laplacian-like matrix used for the attitude synchronization. Simulation results demonstrated the effectiveness of the algorithm. Current work is underway to implement the algorithms using rotorcraft.

Author details

Innocent Okoloko

Address all correspondence to: okoloko@ieee.org

Department of Electrical Engineering, Universidad de Ingenieria y Tecnologia, Lima, Peru

References

[1] Blackwood G, Lay O, Deininger B, Gudim M, Ahmed A, Duren R, Noeckerb C, Barden B. The StarLight mission: A formation-flying stellar interferometer. In: SPIE 4852,

Interferometry in Space; 22 August; Waikoloa, Hawaii. Bellingham WA, USA: SPIE Digital Library; 2002. DOI: 10.1117/12.460942

[2] Beichman CA. NASA's Terrestrial Planet Finder. In: Darwin and Astronomy: The Infrared Space Interferometer; 17-19 November; Stockholm, Sweden. The Netherlands: Noordwijk; 2000. pp. 29-30. DOI: ISSN/ISBN: 03796566

[3] Ahmed A, Alexander J, Boussalis D, Breckenridge W, Macala G, Mesbahi M, Martin MS, Singh G, Wong E. Cassini Control Analysis Book. Pasadena, CA: Jet Propulsion Laboratory, CALTECH Technical Report; 1998. NA p. DOI: NA

[4] Kim Y, Mesbahi M. Quadratically constrained attitude control via semidefinite programming. IEEE Transactions on Automatic Control. 2004;**49**:731-735. DOI: 10.1109/TAC.2004.825959

[5] Wen JT, Kreutz-Delgado K. The attitude control problem. IEEE Transactions on Automatic Control. 1991;**36**(10):1148-1162. DOI: 10.1109/9.90228

[6] Kim Y, Mesbahi M, Singh G, Hadaegh FY. On the convex parameterization of constrained spacecraft reorientation. IEEE Transactions on Aerospace and Electronic Systems. 2010;**46**(3):1097-1109. DOI: 10.1109/TAES.2010.5545176

[7] Okoloko I, Kim Y. Distributed constrained attitude and position control using graph Laplacians. In: ASME Dynamic Systems and Control Conference; 13-15 September; Cambridge, Massachusetts. NY, USA: ASME; 2010. pp. 377-383. DOI: 10.1115/DSCC2010-4036

[8] Okoloko I, Kim Y. Attitude synchronization of multiple spacecraft with cone avoidance constraints. In: IEEE Aerospace Conference; 3-10 March; Big Sky, Montana. NY, USA: IEEEXplore; 2012. pp. 1-10. DOI: 10.1109/AERO.2012.6187119

[9] Okoloko I, Kim Y. Attitude synchronization of multiple spacecraft with cone avoidance constraints. Systems & Control Letters. 2014;**69**:73-79. DOI: 10.1016/j.sysconle.2014.04.008

[10] Bullo F, Murray RM, Sarti A. Control on the sphere and reduced attitude stabilization. In: IFAC Symposium on Nonlinear Control Systems; 25-28 June; Tahoe City, CA. Atlanta GA, USA: Elsevier; 1995. pp. 495-501. DOI: 10.1016/S1474-6670(17)46878-9

[11] Fax AJ. Optimal and cooperative control of vehicle formations [thesis]. Pasadena, CA: PhD Thesis, CALTECH; 2002. 135 p. Available from: thesis.library.caltech.edu/4230/1/Fax_ja_2002.pdf

[12] Ren W. Distributed attitude alignment in spacecraft formation flying. International Journal of Adaptive Control and Signal Processing. 2006;**21**(2–3):95-113. DOI: 10.1002/acs.916

[13] Dimarogonas DV, Tsiotras P, Kyriakopoulos KJ. Leader-follower cooperative attitude control of multiple rigid bodies. Systems and Control Letters. 2009;**58**(6):429-435. DOI: 10.1016/j.sysconle.2009.02.002

[14] Sturm JF. Using SeDuMi 1.02, a Matlab toolbox for optimization over symmetric cones. Optimization Methods and Software. 1998;**11**(12):625-653. DOI: 10.1080/10556789908805766

[15] Lofberg J. Yalmip: A toolbox for modelling and optimization in Matlab. In: IEEE CACSD Conference; 2-4 Sept.; Taipei, Taiwan. NY, USA: IEEEXplore; 2004. pp. 284-289. DOI: 10.1109/CACSD.2004.1393890

[16] Okoloko I. Multi-path planning and multi-body constrained attitude control [dissertation]. Stellenbosch, South Africa: PhD Thesis, Stellenbosch University; 2012. 185 p. Available from: http://hdl.handle.net/10019.1/71905

[17] Kuipers JB. Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace and Virtual Reality. 1st ed. Princeton, NJ: Princeton University Press; 2002. 371 p. ISBN: 13: 978-0691102986

[18] Peng L, Zhao Y, Tian B, Zhang J, Bing-Hong W, Hai-Tao Z, Zhou T. Consensus of self-driven agents with avoidance of collisions. Physical Review. 2009;**79**(E):026113. DOI: 10.1103/PhysRevE.79.026113. Available from: https://journals.aps.org/pre/abstract/10.1103/PhysRevE.79.026113

[19] Olfati-Saber R. Flocking for multi-agent dynamic systems: Algorithms and theory. IEEE Transactions on Automatic Control. 2006;**51**(3):401-420. DOI: 10.1109/TAC.2005.864190

[20] Golub GH, Van Loan CF. Matrix Computations. 3rd ed. Baltimore, MD: Johns Hopkins University Press; 1996. 699 p. ISBN: 13: 978-0801854149

[21] Hughes PC. Spacecraft Attitude Dynamics. 2nd ed. Mineola, NY: Dover Publications Inc; 2004. 592 p. ISBN: 13: 9780486439259

[22] Eberly DH. 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics. 2nd ed. London, UK: Taylor & Francis; 2012. 1015 p. DOI: ISBN: 978-0-12-229063-3

Chapter 4

Search-Based Planning and Replanning in Robotics and Autonomous Systems

An T. Le and Than D. Le

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.71663

Abstract

In this chapter, we present one of the most crucial branches in motion planning: search-based planning and replanning algorithms. This research branch involves two key points: first, representing traverse environment information as discrete graph form, in particular, occupancy grid cost map at arbitrary resolution, and, second, path planning algorithms calculate paths on these graphs from start to goal by propagating cost associated with each vertex in graph. The chapter will guide researcher through the foundation of motion planning concept, the history of search-based path planning and then focus on the evolution of state-of-the-art incremental, heuristic, anytime algorithm families that are currently applied on practical robot rover. The comparison experiment between algorithm families is demonstrated in terms of performance and optimality. The future of search-based path planning and motion planning in general is also discussed.

Keywords: A*, RRT, holonomic path planning, trajectory planning, occupancy map, D* Lite, incremental planning, heuristics planning, ARA*, anytime dynamic A*

1. Introduction

Nowadays, as the rapid advances of computational power together with development of state-of-the-art motion planning (MP) algorithms, autonomous robots can now robustly plan optimal path in narrow configuration space or wide dynamic complex environment with high accuracy and low latency. These recent MP developments have a large impact in medical surgery, animation, expedition and many other disciplines. For instance, RRT [1] algorithm was applied for multi-arm surgical robot in [2]. Expedition robot GDRS XUV was implemented field D* any-angle path planner [3] that enables the robot to optimally move in harsh environment. D* [4] is implemented for Mars Rover prototypes and tactical mobile robots in [5]. Bug algorithms were implemented in multi-robot cooperation scenarios [6].

In general, the problem statement of MP can be generalised as follows: Given the initial defined world space and the robot's configuration space, the MP algorithm must generate a series of consecutive collision-free configurations of the robot that connects start configuration and goal configuration. This series configuration must satisfy any inherent motion or non-motion constraints of the robot.

To cope with a wide range of environment characteristics, MP can be divided into two categories: gross MP and fine MP [7]. The gross MP concerns with the scenarios when world space is much wider than obstacles' size and positional error of the robot, whereas the fine MP solves the planning problems in narrow space that requires high accuracy.

This manuscript presents the development of gross MP algorithm family, in particular search-based planning and replanning paradigm. The foundation concepts of MP, configuration space representations, and the position of mentioned paradigm in MP big picture is presented in Section 2. Section 3 describes historical basis of search-based algorithm family. Section 4 demonstrates the properties and pitfall of D* Lite, which is one of the most crucial algorithms to plan path in dynamic environment. After that, the variants of D* Lite, which improve D* Lite's optimality and performance, are presented. To confirm the improvements, we provide experimental results of recent path planning algorithms and their comparisons in terms of performance and optimality in Section 5. Section 6 will discuss about the future development of MP and provide conclusion.

2. Motion planning concepts

This section will provide an overview of the basic elements that every MP problem must involve. These elements are configuration space of robot and obstacles, environment representation, MP method and search method. The mentioned factors must be analysed consecutively in order to apply suitable MP algorithm family for each scenario.

2.1. Classification of motion planning problems

There still does not exist unified MP algorithm that can robustly solve MP problems in any scenarios such as time optimality, path optimality, moving target, non-holonomic motion, etc. However, with the active recent development of MP, a variety of MP algorithm families are invented to deal with the mentioned scenarios. We will provide detail MP algorithm family classifications based on problem type and therefore demonstrate the location of search-based paradigm in MP.

Figure 1 describes the family tree of MP algorithms based on problem-type classification.

As can be seen, MP with non-holonomic (velocity and kinodynamic) constraints, which is handled by sampling-based paradigm, is still an open research area due to the hardness of transforming high DOF robot and surroundings into configuration space. This configuration space problem has been proved to be NP hard, and computing configuration space operation has

http://dx.doi.org/10.5772/intechopen.71663

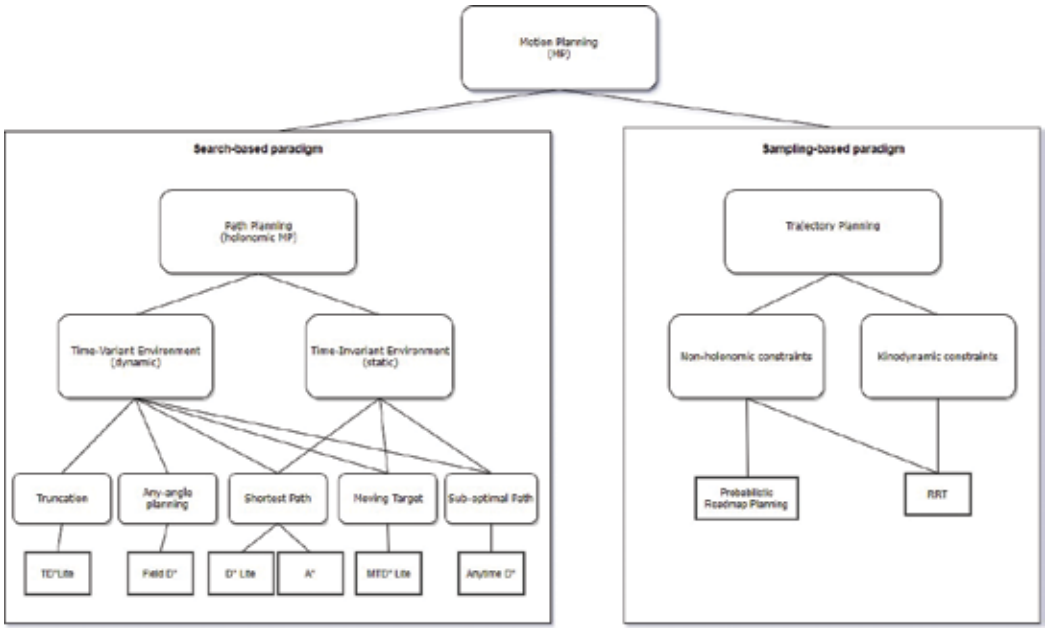

Figure 1. Classification of MP algorithm families based on problem type; the deepest leaves of algorithm tree are representatives for their families.

exponential lower bound [7]. Until recently, the mainstream of non-holonomic MP research is developed based on random rapidly exploring random tree planner (RRT). For example, heuristics property of A* [8] has been applied to RRT for faster trajectory convergence [9]. Fast Marching Square method was developed for non-holonomic car-like robot based on RRT that produces smoother trajectory than RRT [10].

Unlike sampling-based paradigm, search-based paradigm, which represents for path planning algorithms, has a long history of evolution, from basic graph searching to dynamic motion planner with constraints. In this paradigm, robot is treated as point or scalar robot that is able to move in any direction at any time interval. Hence, the configuration obstacle space has the same dimension with the environment, and the generated trajectory is just a path in operating environment. Search-based paradigm is divided into time-invariant and time-variant environment categories. A* is the representative for time-invariant algorithm family; its cost function is incorporated with heuristic property for faster optimal path planning. When dealing with time-variant problem, although we can ensure the optimality and correctness of path solution, we cannot just rerun A* from the point that the robot detects changes in environment due to high latency. To efficiently path replanning in dynamic environment, incremental property is combined with heuristic property to develop D* Lite algorithm; this algorithm is the basis for future development of search-based replanning. Many variants of D* Lite for different MP problems are presented in **Table 1**.

The development of search-based algorithm family is described detail in Section 3 and Section 4.

Problem scenarios	Algorithms
Moving target	MTD* Lite [11]
Fast/suboptimal	Anytime D* [12], truncated D* Lite [13], anytime Truncated D* [14]
Any-angle movement	Field D* [3], incremental Phi* [15]
Performance improvement	D* Lite with Reset [16]

Table 1. Different families of D* Lite variants.

2.2. Problem statement formulation

The general MP problem can be formulated as the following six terms:

1. **State space**: the configuration space of the robot transformed from physical space, W.
2. **Boundary values**: $x_{init} \in W$ and $X_{goal} \subset W$.
3. **Collision detector:** $D: W \rightarrow \{true, false\}$ the function to detect whether the global constraints are satisfied from robot state x; it can output binary or real values.
4. **Input space:** a set U of input, which specifies a complete set of robot operation that affects the state x.
5. **Incremental rules:** a set of rules to transition state $x(t)$ to state $x(t + \Delta t)$ when an operation is input over time interval, $\{u(t_c) | \ t < t_c < t + \Delta t\}$.
6. **Metric:** a real-valued function $\rho: W \times W \rightarrow [0, +\infty)$ that defines the distance between two points in state space W.

General MP is viewed as a search for path (a series of configuration) in state space W that connects start configuration x_{init} to goal configuration region X_{goal}. The robot is incorporated with a set of global constraints (small discrete headings, velocity, balancing, etc.). We denote W_{free} as a set of configuration that satisfies global constraints, and the generated path must be in W_{free}. The incremental rules can be considered as discrete-time response system, and together with input space, it defines possible robot state transitions. Metric can affect heavily to the algorithm's optimality and performance; it indicates the distance between pair of points in topological space. One can construct MP algorithm to deal with specific constraints in certain environment by following these basic terms.

2.3. Environment representation

This section will describe the transformation of world space to state space. This is the first step to formulate a MP algorithm; it creates an operation environment for MP algorithm and a way to represent physical world information as data structure in computer.

2.3.1. Configuration space (C-space) transformation

The world space (physical space) is where the robot and obstacles exist; it is a map of the practical world. However, we cannot apply directly MP algorithm to this space due to the

hardness of representing orientation dimension and other parameters such as motion constraints on computer. Therefore, a C-space is needed, which incorporates all independent parameters that completely define the position of all points on the robot and specifies global constraints of the robot as Cartesian space. **Figure 2** [17] shows a mapping between an effector of 2DOF robot arm and a set of possible two angle parameters that constitutes C-space of the effector.

After computing the C-space, all MP problems are basically reduced to finding a series of configuration that connects start configuration and goal configuration. In other word, the problem is reduced to finding a path for a point robot from start to goal. The number of parameters that defines robot position is the dimension of C-space. The method to compute C-space is mentioned in [7].

For simplicity, to follow the scope of this chapter, we will treat C-space of point robot the same as world space; the reason is that search-based paradigm deals with holonomic MP problem in which the size of robot is neglected compared to operating environment.

2.3.2. Continuous to discrete approximation

After transforming world space to C-space, we still cannot apply search-based algorithms to C-space. The problem is that search-based algorithms like A* or D* Lite work on graph-like structures; hence, applying search-based algorithms on continuous C-space is intractable. However, other MP algorithm families such as sampling based can apply directly to C-space. Unfortunately, the path optimality and performance of sampling-based algorithms are currently worse than state-of-the-art search-based algorithms.

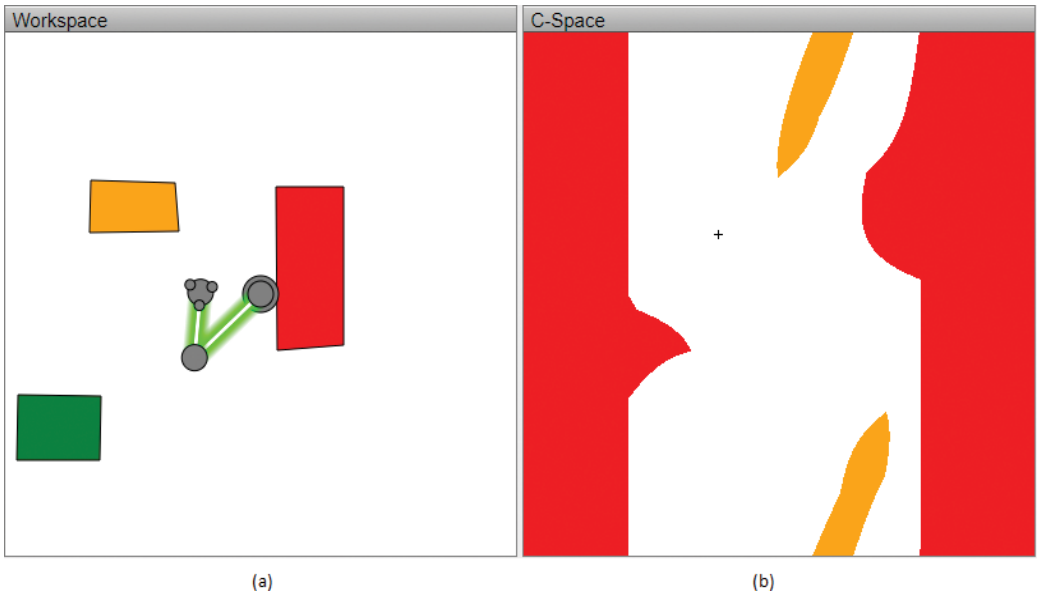

Figure 2. Configuration space of 2DOF robot arm that represents a set of collision-free angles in white and specific object collided in colours (a) Workspace, (b) C-space.

There are two main approaches to discretize C-space into graph-like structure:

- Cell decomposition
- Roadmap

In cell decomposition approach, we divide C-space into eight-connected square grid environment with arbitrary resolution. Then we colour all cells that intersect with obstacle configuration with black, and other free cells are white. **Figure 3** illustrates this approach.

This approximation has limited assumptions on obstacle configuration. Therefore, the approach is used widely in practice. However, there is no concept of path optimality, because we can infinitely divide C-space into smaller squares. It is a trade-off between optimality and computation. Cell decomposition in high dimensions is also expensive; it has exponential growth in PSPACE.

In roadmaps approach, the idea is avoiding scanning the entire C-space by computing an undirected graph with "road" edges that are guaranteed to be collision-free. The main methods of this approach are visibility graph [17] and Voronoi diagrams. The examples of the two methods are demonstrated in **Figure 4**.

As can be seen, this approach generates fewer vertices than cell decomposition approach. Visibility graph method tends to generate with vertices that are the vertices of obstacles; this property leads to finding shortest path. However, the visibility graph's roadmaps are close to obstacles; collision is inevitable due to some movement error. Voronoi diagram solves the problem by generating roadmaps that keep robot as far away as possible from obstacles.

Despite this approach constructs efficiently graph representation for search-based algorithm; it is difficult to compute in higher dimension or non-polygonal environment. The approach

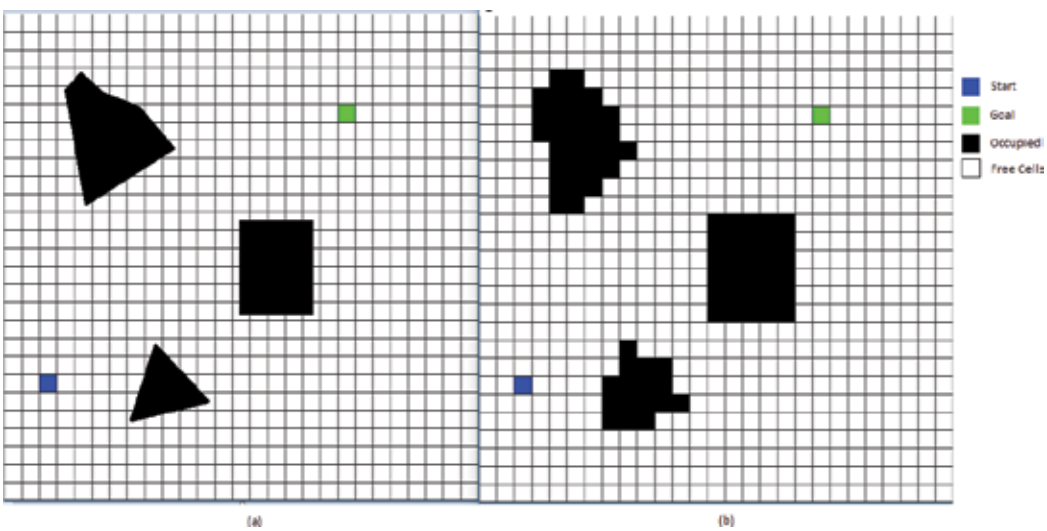

Figure 3. Cell decomposition approach (a) Original Objects, (b) Encoded Objects into cells.

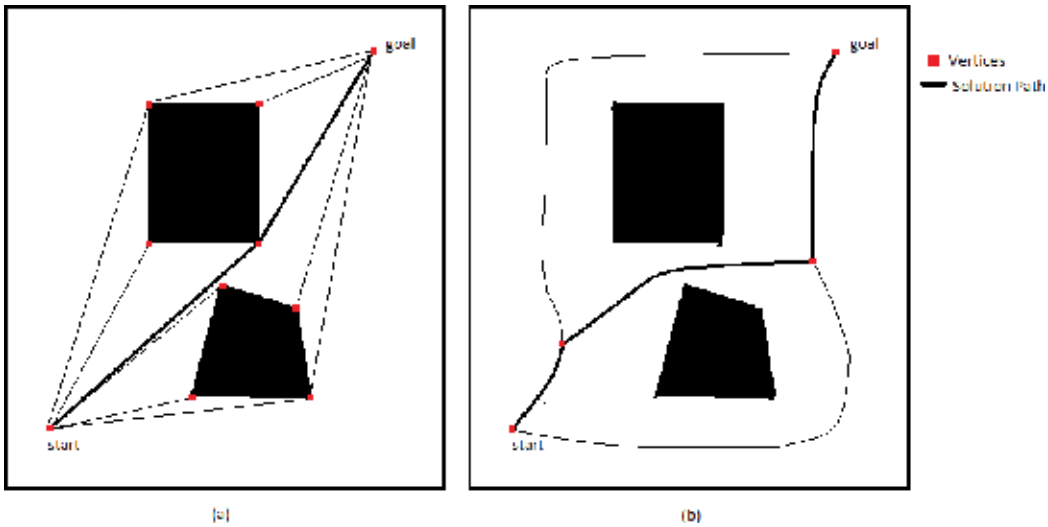

Figure 4. Path topologies of visibility graph and Voronoi diagram methods in roadmap approach (a) Visibility Graph method, (b) Voronoi method.

also can be unstable in dynamic scenarios; small changes in obstacles can lead to large changes in graph.

In the following sections, we use cell decomposition approach for search-based algorithms due to its clarity to describe the operation of search-based algorithms and its feasibility to apply in practice.

3. Search-based planning on time-invariant environment

This section demonstrates one of the most well-known algorithms in graph search family: A*. The A* algorithm's properties are also examined and utilised to use in different cases.

3.1. A* algorithm

There are three main properties of A* [8] that are inherited from historical graph search algorithms:

- **Search tree:** a search tree T, which root is the starting cell, stores expanded cells as branches. This tree is capable to extract path to starting cell from any expanded cell s in the map. A* inherits this tree from breadth-first search algorithm.
- **Uniform cost search:** This property includes a data structure $g(s)$ that stores the cost to travel from starting cell to any cell s in the map, which is formulated as

$$f(s) = g(s), \tag{1}$$

where f(s) is the priority of cell in open list *O*; the smaller the f(s), the higher the priority. The open list *O* handles processing expanding cells, and therefore this property prioritises expanding cells with less cost to travel. A* inherits this property from Dijkstra's algorithm.

- **Heuristic:** a rule to guide expanding search towards goal cell. This rule is formulated:

$$f(s) = h(s), \tag{2}$$

where $h(s)$ is the heuristics function for each cell s that indicates the closeness from cell s to goal. $h(s)$ can be Euclidean distance or Manhattan distance function in this case. In addition, $h(s)$ must satisfy admissible property:

$$h(s) \leq cost(s, s') + h(s'), \tag{3}$$

for any successor s' of s to ensure path optimality. A* inherits this property from greedy best-first search.

Figure 5 illustrates each property of A* when they are applied to search for goal:

The total expanded cells in each algorithm constitutes for their performance (e.g. how many cells are processed before path is found). As can be seen, Dijkstra's algorithm has the worst performance due to lack of guidance to expand search; it just expands uniformly to all directions. Greedy best-first search has the best computation; however, it does not guarantee the shortest path like Dijkstra's algorithm, because its search is trap in local minima shown in the picture. A* has both computation and optimality advantages over these old algorithms by combining uniform cost search rule to guarantee path optimality and heuristic rule of greedy best-first search to guide search process towards goal. Both rules can be combined and formulated as priority function:

$$f(s) = g(s) + h(s). \tag{4}$$

Intuitively, one could think f(s) is an estimated cost to travel from start cell to goal through concerning cell s. Hence, A* expands towards cells that have least cost travel (**Figure 6**, line 11).

The pseudo code for A* is shown in **Figure 6**.

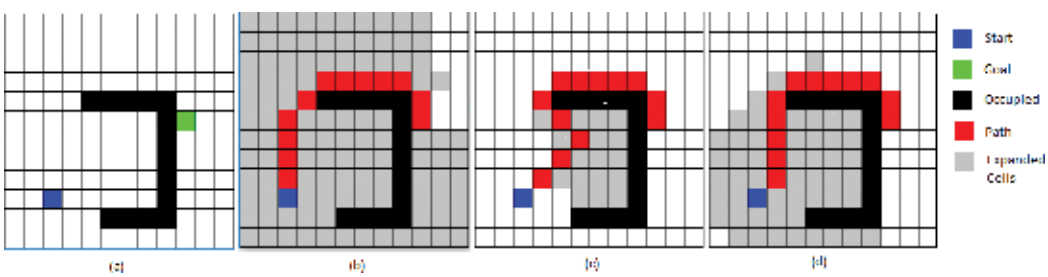

Figure 5. Operation demonstration of properties of A* and A* itself (a) Map, (b) Uniform cost search, (c) Greedy Best-First Search and (d) A*.

Algorithm 1: A*

Input : Graph G, s_{start}, s_{goal}
Output: Search Tree TREE

```
1  OPEN = TREE = ∅
2  Set all nodes g(s) = ∞
3  s_start = 0
4  OPEN.insert(s_start, g(s_start) + h(s_start, s_goal))
5  TREE(s_start) = None
6  while OPEN is not empty OR s is not s_goal do
7  |  s = OPEN.Pop()
8  |  forall s' ∈ Successor(s) do
9  |  |  if g(s) + cost(s, s') < g(s') then
10 |  |  |  g(s') = g(s) + cost(s, s')
11 |  |  |  OPEN.insert(s', g(s') + h(s', s_goal))
12 |  |  |  TREE(s') = s
13 |  |  end
14 |  end
15 end
```

Figure 6. Pseudo code of A* algorithm.

3.2. Anytime A*: path suboptimal bound (ARA*) algorithm

In practice, the performance issue is more critical; time for robot to "think" before making decision is limited. Therefore, a path planner, which has these properties, is essential:

- Quickly producing a suboptimal solution and then gradually improving its solution as time allowed by reusing its previous search effort as much as possible
- Having control over the suboptimal bound and hence indicating a bound of processing time of each search iteration

We introduce the algorithm that is well-suited for this scenario: ARA* [18].

Basically, ARA* is developed from A*; it inherits all intrinsic properties of A*. The idea to quickly plan suboptimal path is derived from inflated heuristics function [18] by a factor ε. The search is greedier to provide solution faster, and the solution is proven to be bounded:

$$g^*(s) \leq g(s) \leq \varepsilon * g^*(s), \tag{5}$$

where $g^*(s)$ is the optimal path cost from start to s.

The pseudo code for ARA* is shown in **Figure 7**.

To understand the behaviours of ARA*, we must keep in mind that ARA* violates admissible property—$h(s) \leq cost(s, s') + h(s')$—for any successor s' of s. ARA* modifies A* $f(s)$ function by inflating heuristics function $h(s)$:

Algorithm 2: ARA*

```
Function Key(s):
    return g(s) + ε * h(s)
Function ImprovePath():
    while Key(s_goal) > min_{s∈OPEN}(Key(s)) do
        s = OPEN.Pop()
        CLOSED.append(s)
        forall s' ∈ Successor(s) do
            if g(s) + cost(s, s') < g(s') then
                g(s') = g(s) + cost(s, s')
                if s' ∉ CLOSED then
                    OPEN.insert(s', Key(s'))
                else
                    INCONS.insert(s')
                end
            end
        end
    end
Function Main():
    Set all nodes g(s) = ∞
    OPEN = CLOSED = INCONS = ∅
    g(s_start) = 0
    OPEN.insert(s_start, Key(s_start))
    ImprovePath()
    Choose initial sub-optimal bound ε
    Publish ε
    while ε > 1 do
        Decrease ε
        Move nodes in INCONS to OPEN
        Update all priority values in OPEN according to Key(s)
        CLOSED = ∅
        ImprovePath()
        Publish ε
    end
```

Figure 7. Pseudo code of ARA* algorithm.

$$f(s) = g(s) + \varepsilon * h(s). \tag{6}$$

Hence, the computed path is no longer optimal. Moreover, each search iteration is no longer guaranteed to expand searching each cell at most once like A* due to decreasing ε. However, to maintain efficiency and ensure suboptimal bound, ARA* introduces INCONS list to store local inconsistent cells as specified function:

$$g(s') > \min_{s'' \in pred(s')}\left(cost(s', s'') + g(s'')\right), \tag{7}$$

(**Figure 7**, line 13) that already are expanded once and processes these cells in the next search iteration.

In general, ARA* executes consecutive search iterations with decreasing suboptimal bound; each search does not recalculate consistent cells from previous search. Therefore, the path improvement process is efficient. Theoretical properties of ARA* is described in [18].

4. Search-based replanning on time-varying environment

In real-world application, there is often a scenario that the robot initially does not know a priori information about its surroundings. We cannot encode the world space information each time the robot runs, because it is expensive, tedious, and infeasible due to rapid changes in practice. To maintain collision-free path, one can naively rerun A* to replan the shortest path from the point that the robot detects changes. However, this naïve approach will waste computation by reprocessing cells that are irrelevant to compute a new path and hence increase idle time between each search. This section will demonstrate search-based algorithms to solve mentioned problem in time-variant environment.

4.1. Incremental heuristic algorithm: D* Lite algorithm

4.1.1. D Lite algorithm*

In goal-directed navigation task, with cell decomposition approximation, the robot always observes a limited range of eight connected grids. The robot is able to move in eight directions with cost one, and it assumes that unknown cells are traversable. The robot follows the initial calculated path to goal and encounters blockage cells; it must be able to process only cells that are relevant to compute the new path. The challenge is to find these relevant cells. **Figure 8** illustrates this idea.

Note that grey cells (in **Figure 8**) are expanded cells to compute initial path or new path when robot detects blockage cell in purple at position yellow cell. Darker grey cells are processed multiple times. As can be seen, total expanded cells in replanning process of D* Lite is 61, whereas expanded cells of rerunning A* are 75.

D* Lite [19] is developed directly from Lifelong Planning A* (LPA*) [20] for applying on mobile robot, which is a combination of Dynamic SWSF-FP [21] and A* [8]. Therefore, D* Lite possesses these properties:

- Reverse search: Unlike A*, D* Lite expands its search from goal; $h(s)$ now indicates the closeness from cell s to start cell. $g(s)$ now also stores estimated distance from goal. After searching is finished, the path from start to goal is generated by iteratively moving from cell s towards neighbour cells s' that have the lowest sum $g(s) + cost(s, s')$ in greedy style.
- Heuristics: D* Lite inherits this property from A* with admissible rule. Thus, D* Lite maintains path optimality by expanding heuristically towards start cell.

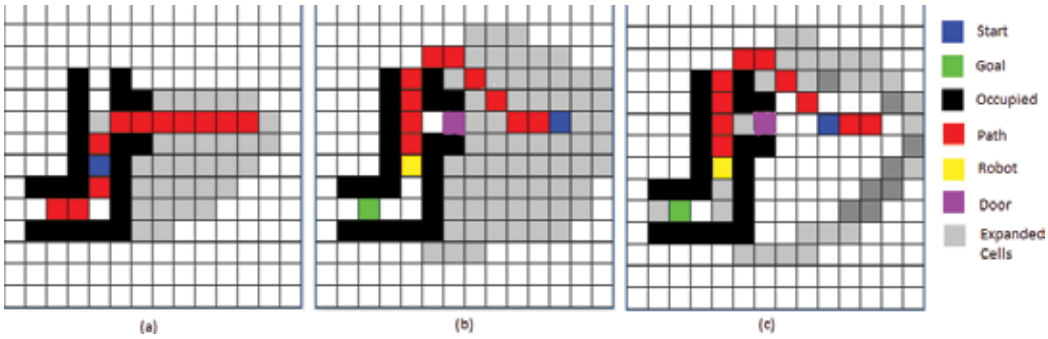

Figure 8. MP simulation on grid environment (a) Initial path, (b) Reset A* and (c) D* Lite.

- Incremental: D* Lite inherits incremental search property from Dynamic SWSF-FP; it reuses information from previous search to repair path in a series of similar searches, which is much efficient than calculating path from scratch.

The pseudo code for D* Lite is shown in **Figure 9**.

In general, the pseudo code of D* Lite maintains three invariants:

- Invariant 1: $rhs(s) = \begin{cases} 0 & if\ s = s_{start} \\ min_{s' \in Pred(s)}(g(s') + cost(s, s')) & otherwise \end{cases}$.
- Invariant 2: OPEN list contains exactly only local inconsistent cells $g(s) \neq rhs(s)$.
- Invariant 3: Priority value of cells in OPEN list is equal to its $Key(s)$.

At the first run, D* Lite is exactly like A*. It guarantees to expand cells at most twice in each search routine due to the concept of one-step look-ahead estimated goal distance r $hs(s)$ that is inherited from LPA*. r $hs(s)$ leads to the terms of over-consistent cell $g(s) > rhs(s)$ and under-consistent cell $g(s) < rhs(s)$. Intuitively, these concepts help propagating the inconsistency of cells to their neighbours. To maintain Invariants 1 and 2, ComputePath() function updates rhs-values of changed cells, checks their consistency and decides their membership of OPEN list accordingly. Invariant 3 is maintained by updating the OPEN list keys while expanding (**Figure 9**, lines 17–18). ComputePath() stops when the smallest key of OPEN list is less than $Key(s_{start})$ or s_{start} is consistent; this criteria indicates that cell expansion has reached target s_{start}. Theorems of D* Lite are described detail in [19].

4.1.2. Pitfall of D Lite*

Despite being an effective replanner for dynamic environment, D* Lite does have a big pitfall for certain circumstances. In fact, D* Lite is designed to be implemented in mobile robot with range sensors, in which the environment changes are perceived near the robot (the starting cell). In other word, the changes occurred at the perimeter of expansion. Therefore, D* Lite just propagates inconsistencies in a small area near the search front; the replanning process is efficient. However, the problem arises when we combine other sensors (e.g. UAV, satellite,

Algorithm 3: D* Lite

```
1  Function Key(s):
2      return [min(g(s), rhs(s)) + h(s_start, s) + k_m; min(g(s), rhs(s))]
3  Function UpdateVertex(s):
4      if s ≠ s_goal then
5          rhs(s) = min_{s'∈Succ(s)}(cost(s, s') + g(s'))
6      end
7      if s ∈ OPEN then
8          OPEN.remove(s)
9      end
10     if g(s) ≠ rhs(s) then
11         OPEN.insert(s, Key(s))
12     end
13 Function ComputePath():
14     while OPEN.TopKey() < Key(s_start) OR rhs(s_start) ≠ g(s_start) do
15         k_old = OPEN.TopKey()
16         s = OPEN.Pop()
17         if k_old < Key(s) then
18             OPEN.insert(s, Key(s))
19         else if g(s) > rhs(s) then
20             g(s) = rhs(s)
21             forall s' ∈ Pred(s) do
22                 UpdateVertex(s')
23             end
24         else
25             g(s) = ∞
26             forall s' ∈ Pred(s) ∪ {s} do
27                 UpdateVertex(s')
28             end
29     end
30 Function Main():
31     forall s ∈ S do
32         rhs(s) = g(s) = ∞
33     end
34     s_last = s_start
35     OPEN = ∅
36     rhs(s_goal) = 0; k_m = 0
37     OPEN.insert(s_goal, Key(s_goal))
38     ComputePath()
39     while s_start ≠ s_goal do
40         s_start = argmin_{s'∈Succ(s_start)}(cost(s_start, s')+ g(s'))
41         Move to s_start
42         Scan for cell changes in environment (e.g. sensor ranges)
43         if Cell changes detected then
44             k_m = k_m + h(s_last, s_start)
45             s_last = s_start
46             forall s ∈ CHANGES do
47                 Update cell s state
48                 forall s' ∈ Pred(s) ∪ {s} do
                       UpdateVertex(s')
                   end
               end
               ComputePath()
           end
       end
```

Figure 9. Pseudo code of D* Lite algorithm.

etc.) to detect environment changes in further area near the goal. Intuitively, we can imagine a valley where $g(s)$ of each cell substitutes for its height; the goal cell has the lowest height (the bottom of the valley), and robot position (start cell) is always at valley's edge. Suddenly, there is a change in height near the goal; D* Lite has to give enormous effort to correct the continuity of the valley slope from the bottom to the surface. Because of the overhead of storing

g-value information, the correction effort now is more expensive than starting the search from scratch. This is a big limitation for multi-sensor-based robot system; the problem also makes D* Lite unreliable in high-dimensional state space.

This behaviour leads us to a problem statement: The location of environment changes with respect to goal position makes an enormous difference to efficiency of D* Lite. This problem is also addressed by the author of D* Lite in [12] as open question. In other papers, mathematical approach is used to study this pitfall in [16]. Unfortunately, the problem is still not solved thoroughly; however, there are approaches to partly overcome this pitfall in certain situation that will be presented in the following section.

4.2. Performance improvements

This section describes variants of D* Lite that partially solves the mentioned pitfall of D* Lite. Hence, these state-of-the-art algorithms improve the computation factor of D* Lite.

4.2.1. Anytime dynamic A (AD*) algorithm*

As one of the prominent properties of D* Lite, it maintains the optimality of solution paths. However, in real-world application, optimal paths are difficult to calculate due to the complexity and uncertainty of environment within available time; the paths are also quickly to become out of date because of dynamic surroundings. Moreover, the state space, which encodes global motion constraints of the robot, tends to be high dimensions. With these difficulties, D* Lite becomes unreliable when implementing in real robot.

Anytime Dynamic A* (AD*) [12], which is a combination of D* Lite and ARA* algorithm, is a trade-off between computation and path optimality. It sacrifices the shortest path to quickly generate suboptimal solution to cope with imperfect information and dynamic environment. AD* inherits these properties from its parent algorithms:

- **Anytime:** AD* uses inflated heuristic function to increase the greedy factor that expands aggressively towards goal while still maintaining path suboptimal bounds ε. In addition, AD* also is capable to reuse information from previous search to improve its path. This property is inherited from ARA* to cope with complex planning scenario.
- **Incremental heuristic:** AD* has the ability to efficiently identify relevant cells that contribute to replan a new path when environment changes are detected. However, the heuristic function in this property is not inflated to guarantee the suboptimal bound in replanning process. This property is inherited from D* Lite to cope with dynamic environment.

In general, AD* executes a series of similar searches with decreasing suboptimal bounds to generate a series of paths with improved bounds. As environment changes are detected, the locally inconsistent cells are placed in OPEN list with uninflated heuristic keys, and AD* processes these cells to correct the outdated path. The pseudo code for AD* is shown in **Figure 10**.

http://dx.doi.org/10.5772/intechopen.71663

Algorithm 4: AD*

```
1  Function Key(s):
2      if g(s) > rhs(s) then
3          return [rhs(s)+ε*h(s_start, s); rhs(s)];
4      else
5          return [g(s) + h(s_start, s); g(s)]
6  Function UpdateVertex(s):
7      if s ≠ s_goal then
8          rhs(s) = min_{s'∈Succ(s)}(cost(s, s') + g(s'))
9      end
10     if s ∈ OPEN then
11         OPEN.remove(s)
12     end
13     if g(s) ≠ rhs(s) then
14         if s ∉ CLOSED then
15             OPEN.insert(s, Key(s))
16         else
17             INCONS.insert(s)
18     end
19 Function ComputeOrImprovePath():
20     while OPEN.TopKey() < Key(s_start) OR rhs(s_start) ≠ g(s_start) do
21         s = OPEN.Pop()
22         if g(s) > rhs(s) then
23             g(s) = rhs(s)
24             CLOSED.insert(s)
25             forall s' ∈ Pred(s) do
26                 UpdateVertex(s')
27             end
28         else
29             g(s) = ∞
30             forall s' ∈ Pred(s) ∪ {s} do
31                 UpdateVertex(s')
32             end
33     end
34 Function Main():
35     forall s ∈ S do
36         rhs(s) = g(s) = ∞
37     end
38     OPEN = CLOSED = INCONS = ∅
39     rhs(s_goal) = 0
40     OPEN.insert(s_goal, Key(s_goal))
41     ComputeOrImprovePath()
42     Choose initial sub-optimal bound ε and publish ε
43     while True do
44         Scan for cell changes in environment (e.g. sensor ranges)
45         if Cell changes detected then
46             forall s ∈ CHANGES do
47                 Update cell s state
48                 forall s' ∈ Pred(s) ∪ {s} do
49                     UpdateVertex(s')
50                 end
51             end
52         end
53         if Significant cell changes detected then
54             Increase ε or replan from scratch
55         else if ε > 1 then
56             Descrease ε
57         Move nodes in INCONS to OPEN and set CLOSED = ∅
58         Update all priority values in OPEN according to Key(s)
59         ComputeOrImprovePath()
60         Publish ε
61     end
```

Figure 10. Pseudo code of AD* algorithm.

At first, we set the suboptimal bound ε to be large enough in order to generate solution quickly (**Figure 10**, line 42). Unless environment changes are detected, AD* iteratively decrease suboptimal bound ε to improve the solution as time allowed (**Figure 10**, lines 55–60); this phase is exactly the same with ARA*.

When changes are perceived, the suboptimal bound of solution is no longer guaranteed, especially the under-consistent cells, due to the fact that there could have shorter paths exist. Therefore, these under-consistent cells are placed in OPEN list with uninflated heuristic to ensure these cells propagate their inconsistences to neighbours first when ComputeOrImprovePath() function is called (**Figure 10**, lines 5 and 45–50). However, because of this effect, many under-consistent cells quickly rise to the top of OPEN list; they usually do not contribute to calculate new path in practice (e.g. objects cause under-consistent changes like human movement, etc.). Hence, AD* tends to slow down when the path is near optimal. Theorems of AD* are described in detail in [12].

When "significant cell changes are detected" (**Figure 10**, line 53), there is a high chance that the problem of D* Lite occurs and the search tree may be corrupted heavily; the replanning process now is expensive; we can increase suboptimal bound ε to speed up the correction effort or start a new search from scratch. The problem is how to estimate "significant cell changes". This algorithm does not solve the mentioned problem of D* Lite completely; it is just a trade-off between performance and path optimality. However, AD* performs quickly in large-scaled map compared to other algorithms, in which the robot has significant time to iteratively repair its path, and thus overall path is near optimal.

4.2.2. D Lite with reset algorithm*

Unlike AD*, D* Lite with Reset (D*LR) [16] partially solves the problem D* Lite while still maintaining path optimality. The idea of D*LR is simple; it decides flushing previous search data and starts searching from scratch when the replanning process is expensive.

D*LR is a variant of D* Lite; it inherits all the properties of D* Lite. The main contribution of D*LR is that it proposes two criteria to decide whether to incrementally replan path or calculate a fresh path using A* at the position the robot detects changes. Let total traversed cell is N_T; total cell of path that exists between consecutive detection incidents is N_P, and the remaining path count is $N_R = N_P - N_T$; the criteria are:

- **Ratio of traversed length:** The criteria measure how many percentages of the path the robot has moved between two consecutive positions that the robot detects environment changes. The ratio $\frac{N_T}{N_P}$ is then compared with a threshold:

$$\frac{N_T}{N_P} < \propto . \tag{8}$$

If the ratio is greater than **threshold** $\propto$ when the robot perceives changes, it triggers the reset routine and starts searching from scratch. Intuitively, the position that robot detects changes is nearer the goal, the more likely replanning process is expensive.

- **Linear heuristic distance:** The criteria measure the complexity of the remaining path count N_R between consecutive detection incidents. The method to measure the complexity is to use inflated heuristic function:

$$N_R > \varepsilon * h(s, s_{goal}), \tag{9}$$

where s is the current position of the robot. If $N_R \leq \varepsilon * h(s, s_{goal})$, then the remaining path is simple enough; there is a chance that the new path is much more complex. Hence, the algorithm must plan over from scratch.

As can be seen, these criteria use only path information between consecutive detection incidents in order to estimate the amount of computation of replanning process comparing with planning over from scratch. The reason is that it is hard to predict propagation behaviour of OPEN list, because the state space is only partially known. Moreover, these criteria only work in high cluttered and complex environment, where environment changes usually block initial path and the new path is likely to be much longer than initial path. The pseudo code of D*LR is presented in **Figure 11**.

The proposed criteria of D*LR are not robust due to extensively replying on environmental assumptions. However, the algorithm can be improved if criteria that can robustly estimate computation of replanning process in any kind of environment are applied. If criteria are robust, D*LR performance of each iteration is bounded by the complexity of A*:

$$O(|V|) = O(|E|) = O(b^d) \tag{10}$$

4.3. Optimality improvements: Field D* algorithm

Although cell decomposition approximation is widely used to discretize C-space for search-based algorithms due to its robustness (no prior environmental assumptions), this approximation intrinsically prevents search-based algorithm to produce optimal path. The search-based algorithms just allow to transition between cell centres, thus restricting robot traverse directions to increment of $\frac{\pi}{4}$. Moreover, the produced path involves many sharp turns and jerky segments in large map that makes robot difficult to move.

There were many approaches to cope with this problem. For instance, post-processing method that finds the furthest point P along the solution path for which a straight line path from P to robot position is collision-free and replaces the original path to P with this straight line. However, this method sometimes does not work and increases the path cost. Another approach is fast marching method [22]; this method incorporates interpolation step in planning step to produce low-cost interpolated path. Nonetheless, this method assumes that transition cost between grid cells is constant and does not have heuristic property like A*; hence it is not applicable to outdoor environment, which requires fast path generating and non-uniform cost grid.

To incorporate incremental heuristic property of D* Lite, the authors of Field D* [3] embed linear interpolation method to the replanning process to generate "any-angle" optimal path

Algorithm 5: D* Lite with Reset

```
1  Function Initialize():
2      forall s ∈ S do
3          rhs(s) = g(s) = ∞
4      end
5      OPEN = ∅
6      rhs(s_goal) = 0; k_m = 0
7      OPEN.insert(s_goal, Key(s_goal))
8  Function Main():
9      s_last = s_start
10     Initialize()
11     ComputePath()
12     Count length of path N_P and N_T = 0
13     while s_start ≠ s_goal do
14         s_start = argmin_{s'∈Succ(s_start)}(cost(s_start, s') + g(s'))
15         Move to s_start and N_T = N_T + 1
16         Scan for cell changes in environment (e.g. sensor ranges)
17         if Cell changes detected then
18             if Reset Criteria is satisfied then
19                 Initialize()
20             else
21                 k_m = k_m + h(s_last, s_start)
22                 forall s ∈ CHANGES do
23                     Update cell s state
24                     forall s' ∈ Pred(s) ∪ {s} do
25                         UpdateVertex(s')
26                     end
27                 end
28             s_last = s_start
29             ComputePath()
30             Count length of path N_P and N_T = 0
31         end
32     end
```

Figure 11. Pseudo code of D*LR. Other functions such as Key(), UpdateVertex() and ComputePath() are the same with D* Lite and thus are not presented.

that overcomes grid limitation in dynamic environment. The root cause of restriction of path optimality is the rule to transition between cell centres; the idea of Field D* to solve this problem is to remap state space graph vertices to the corner of each cell (see **Figure 12**). The nodes s can be considered as sample points of continuous cost field, where the optimal path must pass one of the edges $\{\overrightarrow{s_1 s_2}, \overrightarrow{s_2 s_3}, \overrightarrow{s_3 s_4}, \overrightarrow{s_4 s_5}, \overrightarrow{s_5 s_6}, \overrightarrow{s_6 s_7}, \overrightarrow{s_7 s_8}, \overrightarrow{s_8 s_1}\}$ that connects consecutive neighbours of s; the edge is $\overline{s_1 s_2}$ in the picture's case.

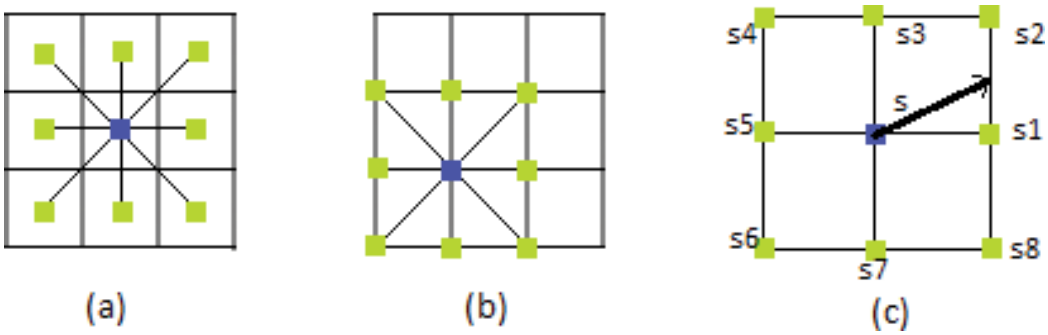

Figure 12. Remapping state space graph vertices from cell centres to cell corners (a) Center Vertices, (b) Corner Vertices and (c) Optimal Path intersected $\longrightarrow s_1 s_2$.

In this case, the edge, which resides on the boundary of two cells, has the edge cost equal to the minimum of cost of the two cells. Field D* use linear interpolation to compute approximately the cost of any point s_y on the edge $\overrightarrow{s_1 s_2}$ by using the path cost (cost from the node to goal) $g(s_1)$ and $g(s_2)$:

$$g(s_y) = yg(s_2) + (1-y)g(s_1), \tag{11}$$

where y is the distance from s_1 to s_y (**Figure 13**). Given the centre cell cost c and bottom cell cost b, we can compute the path cost of s using edge $\overrightarrow{s_1 s_2}$ as

$$g(s) = \min_{x,y} \left(bx + c\sqrt{(1-x)^2 + y^2} + g(s_2)y + g(s_1)(1-y) \right) \tag{12}$$

where x is the distance travel along the bottom edge from s before cutting through the centre cell to reach the right edge at the point s_y a distance y from s_1. (see **Figure 13**).

The interpretation from formulas (4) into ComputeCost() function is described in detail in [3]. This optimization approach can be plugged in any dynamic planner by replacing standard cost function between cell centres by function ComputeCost(). In addition, due to remapping

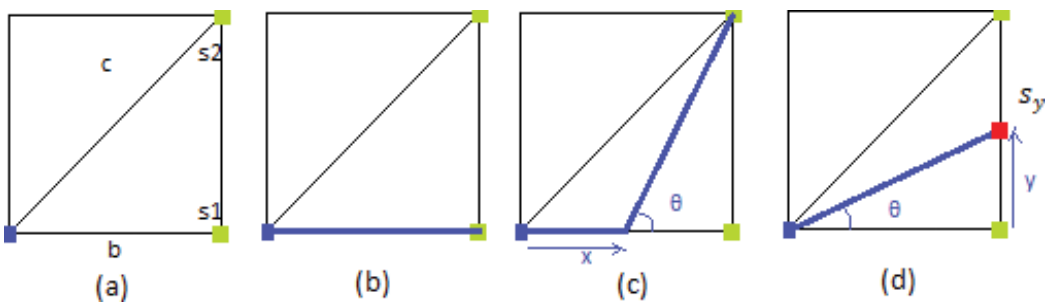

Figure 13. Linear interpolation process to compute path cost of s using edge $\longrightarrow s_1 s_2$. The subfigures illustrate possible optimal path cost (a) Calculate g(s) based on s_1, s_2 (b) Case $g(s_1) < g(s_2)$ (c) Case $g(s_1) > g(s_2)$ and (d) Case path costs that pass arbitrary point.

graph vertices into cell corners, we also need to change finding cell centre neighbours to a pair of corner nodes as illustrated in **Figure 13**. Once the path costs of necessary nodes are computed, the path is generated by starting from the initial node and iteratively finds, using linear interpolation, the optimal node on the neighbor cell boundary to move next. The pseudo code of Field D* and modifications in red colour are shown in **Figure 14**. Note that the differences

Algorithm 6: Field D*

```
1  Function ComputeCost():
2      if s_a is diagonal neighbor of s then
3          s_1 = s_b; s_2 = s_a
4      else
5          s_1 = s_a; s_2 = s_b
6      Set c is traversal cost of cell with corners s, s_1, s_2
7      Set b is traversal cost of cell with corners s, s_1 but not s_2
8      if min(c, b) = ∞ then
9          v_s = ∞
10     else if g(s_1) ≤ g(s_2) then
11         v_s = min(c, b) + g(s_1)
12     else
13         f = g(s_1) − g(s_2)
14         if f ≤ b then
15             if c ≤ f then
16                 v_s = c√2 + g(s_2)
17             else
18                 y = min(f/√(c²−f²), 1)
19                 v_s = c√(1+y²) + f(1 − y) + g(s_2)
20         else
21             if c ≤ b then
22                 v_s = c√2 + g(s_2)
23             else
24                 x = 1 − min(b/√(c²−b²), 1)
25                 v_s = c√(1+(1−x)²) + bx + g(s_2)
26     return v_s
27 Function UpdateVertex(s):
28     if s ≠ s_goal then
29         rhs(s) = min_{(s',s'')∈connbrs(s)}(ComputeCost(s, s', s'')
30     end
31     if s ∈ OPEN then
32         OPEN.remove(s)
33     end
34     if g(s) ≠ rhs(s) then
35         OPEN.insert(s, Key(s))
36     end
37 Function Main():
38     forall s ∈ S do
39         rhs(s) = g(s) = ∞
40     end
41     s_last = s_start
42     OPEN = ∅
43     rhs(s_goal) = 0; k_m = 0
44     OPEN.insert(s_goal, Key(s_goal))
45     ComputePath()
46     while s_start ≠ s_goal do
47         Move s_start along interpolated path.
48         Scan for cell changes in environment (e.g. sensor ranges)
49         if Cell changes x detected then
50             k_m = k_m + h(s_last, s_start)
51             s_last = s_start
52             forall x ∈ CHANGES do
53                 Update cell x state
54                 forall Node s' on a corner of x do
55                     UpdateVertex(s')
56                 end
57             end
58             ComputePath()
59         end
60     end
```

Figure 14. Pseudo code of Field D*.

between D* Lite and Field D* are highlighted in red. The function Key(), ComputePath() are the same as D* Lite and thus is not presented. This pseudo code is a basic version of Field D*; optimised versions are presented in [3].

Field D* inherits all properties of D* Lite; it combines linear interpolation method to compute path from any point inside cell, not just corners or cell edges. This feature is crucial for robot to get back on track if the actuator execution is faulty. Moreover, Field D* is not subjected to direction restriction; hence, it produces much shorter and smoother path.

5. Experimentation

In this section, using our path planning framework, we demonstrate the evaluation comparison between algorithms in search-based family in terms of performance and path optimality.

5.1. Evaluation method

To visualise the evolution in computation of search-based algorithms, we compare the replanning computation of D* Lite, Anytime Dynamic A* and D* Lite with Reset. The purpose of the comparison is to demonstrate the performance improvements of D* Lite variants in order to apply on robot that operates in complex and dynamic environment. However, since the planning time depends on the implementation and machine configuration, we therefore choose the amount of cell expansion in each replanning iteration of search-based algorithm to be standard performance measurement of the mentioned algorithms. This method is independent on machine specifics and actual implementation and therefore firmly accurately shows the enhancement of this evaluation. The path solution ratio between AD* with different ε suboptimal bound and optimal path of other algorithms is also measured to visualise the trade-off between optimality and computation.

The experiments are conducted on our 2D simulation engine. The state space is a 2D grid cell with uniform resolution [23]. The conceptual robot in this simulation has two-cell-unit range and its own known grid map to detect environment changes (unblocked cell to blocked cell and vice versa) as it moves along the initial path (see **Figure 15**).

5.2. Evaluation results

We evaluate the performance and path solution of search-based algorithms in two scenarios: partially known and unknown 2D grid environment with uniform resolution. The total expanded cells are averaged based on total replanning processes on each simulation instance, with 95% confident. The path solution of each algorithm is counted as the total cells that the robot has traversed from corner to corner of the map. We decrease the suboptimal bound of AD* for 0.1 per step the robot travels until the suboptimal bound reaches 1.0 (optimal path).

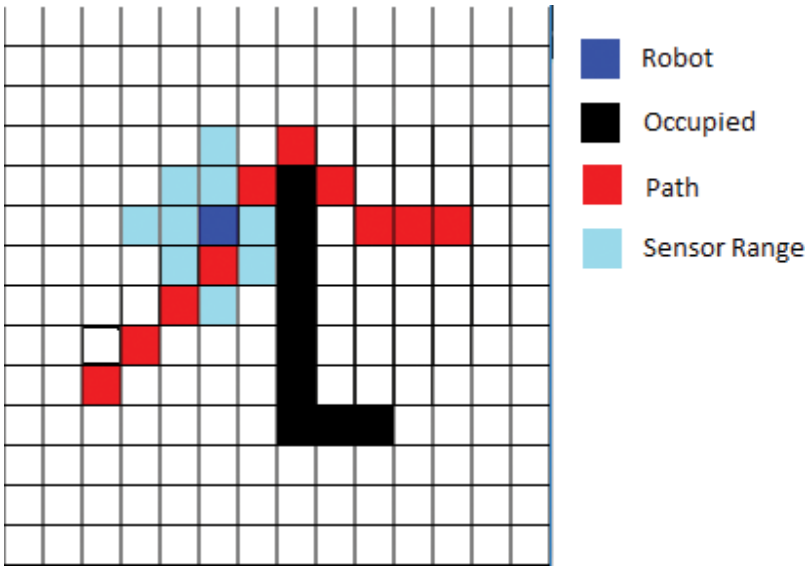

Figure 15. Simulated environment on our framework.

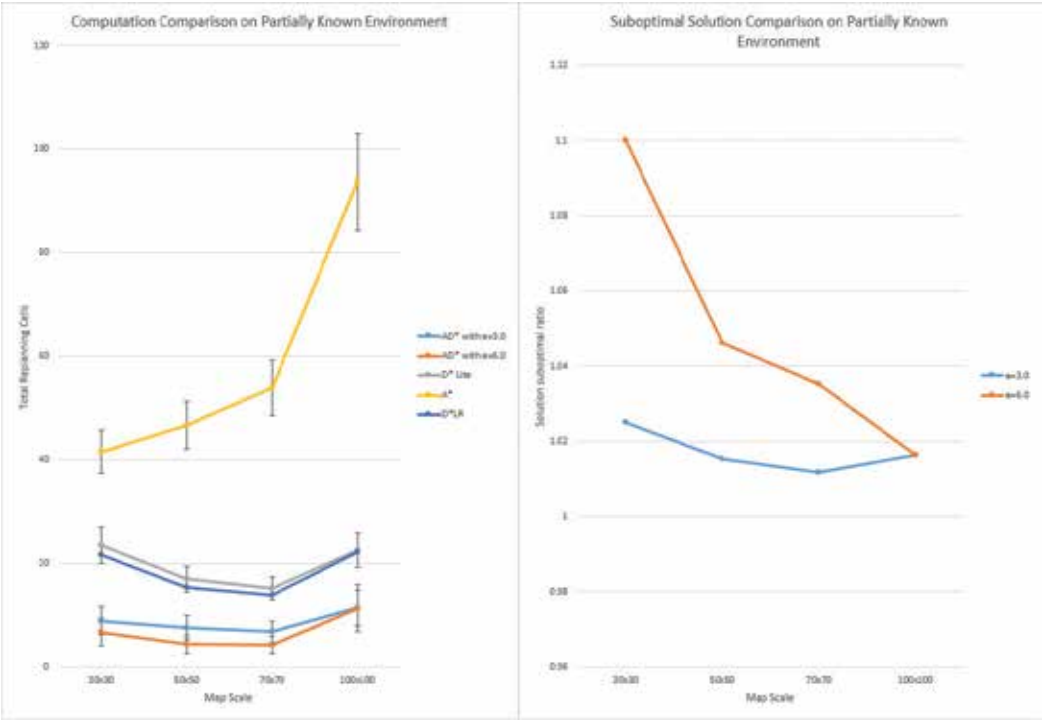

Figure 16. Comparison between search-based algorithms on partially known environment with increasing map scale in terms of computation and path solution.

Figure 16 shows the speedup result throughout the evolution from Replanning A* to AD* with different ε suboptimal bounds as well as the trade-off of AD*. The environment is initially generated randomly obstacles that occupy 25% of the map. The initial map is then input to the robot. While the robot is moving, we randomly change the cell states that are 15% of the map, thus forcing the robot to replan its path whenever it detects environmental changes.

As can be seen, AD* has the highest performance that has least total expanded cells in replanning process; the higher the suboptimal bound, the better the performance. The reason is AD* is inflated its heuristic function to make it greedier in expanding cells towards goal. It is interesting that path solution of AD* is not much longer than optimal path. As the scale of map is increasing, the path between map corner is longer to travel, and thus, the robot is given enough time to improve its solution (path ratio with $\varepsilon = 6.0$ is gradually converged to the one $\varepsilon = 3.0$ at 1.016).

D*LR slightly improves the performance of D* Lite; it is because D*LR relies on computation differences between Replanning A* and D* Lite. In fact, the pitfall of D* Lite rarely happens in scenarios that the robot detects changes near its position. Replanning A* does not have incremental property and thus uses the highest computation.

The data confirms the fact that AD*, in average throughout the increasing map scale, improves 125% and 194% performance compared to D* Lite with $\varepsilon = 3.0$ and $\varepsilon = 6.0$, respectively. The path produces by AD* only 1% longer than optimal path in average.

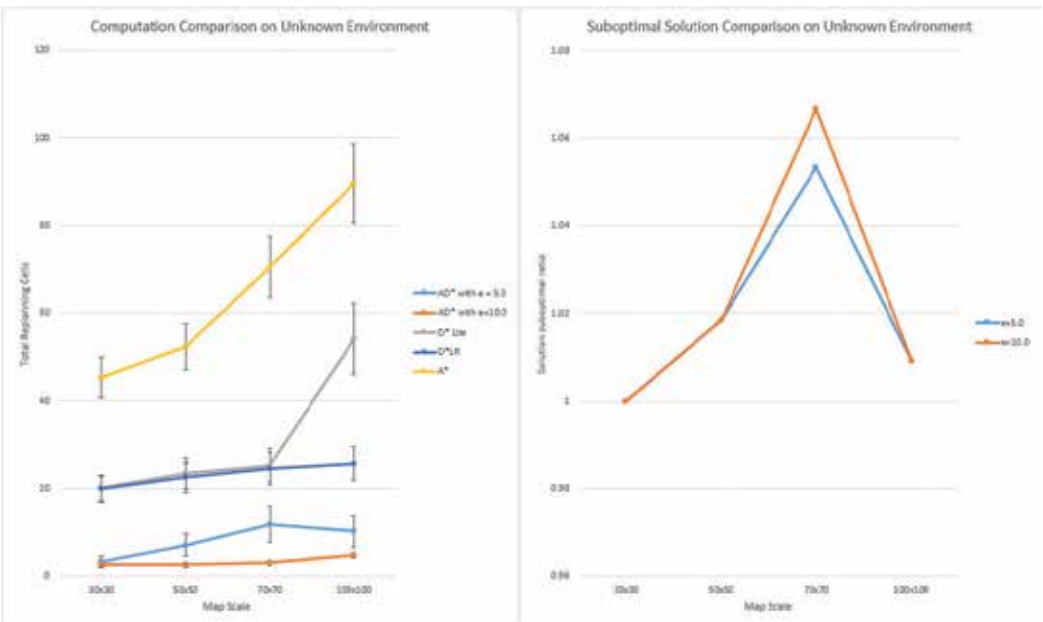

Figure 17. Comparison between search-based algorithms on unknown environment with increasing map scale in terms of computation and path solution.

Figure 17 describes the evaluation case on unknown environment. The environment is initially generated with random obstacles that occupy 15% of the map. The robot does not know the initial conditions; it will replan its path whenever it detects obstacles that do not exist in its map.

For unknown environment scenario, D*LR performs significantly better than D* Lite as increasing map scale. The reason is that if the replanned path is much longer than the initial path, which is the common case in unknown environment, the replanning process of D* Lite is also expensive. AD* still has the least computation compared to old search-based algorithm; it reduces drastically the computation of D* Lite with 845% better performance, in the case $\varepsilon = 10.0$, while still maintains good path solution.

6. Conclusion

In practice, motion planning algorithms can be implemented on top of navigation layer such as simultaneous localization and mapping (SLAM) for autonomous robot. While navigation layer enables the robot to perceive surrounding information and its position relative to the surroundings, motion planning layer gives the robot abilities to plan a path in surrounding environment and make decision to avoid obstacles. Because of that fact, navigation and motion planning are always paired up to enable autonomous robot to operate in dynamic and complex environment.

This chapter is a guide to comprehend the foundation of motion planning, in particular, search-based path planning algorithms. In this chapter, we present the steps to develop and formulate a motion planning problem. We also describe the evolution branches of motion planning and then focus on the development of search-based algorithm family. Each algorithm in search-based family is invented to cope with increasing demands in performance or solution quality, for the robot to operate in more complex scenarios. To reinforce the revolution statement of state-of-the-art search-based algorithms, we provide a computation and optimality comparison between search-based algorithms on partially known and unknown environment. Based on the data, we conclude that Anytime Dynamic A* is the most suitable algorithm that enables the robot to operate in cluttered and fast changing scenario.

Until recently, the mainstream of motion planning development is to enhance the performance of search-based algorithm and their solution optimality by modifying cell decomposition method. There are signals that the trajectory planning paradigm is starting to be active research field after being frozen for a decade. We expect that the future development of trajectory planning will robustly incorporate motion constraints with higher optimality and better computation. The ultimate goal of motion planning field is giving robot spatial decision planning converging to human ability.

Author details

An T. Le[1]* and Than D. Le[2]

*Address all correspondence to: eeit2015_an.lt@student.vgu.edu.vn

1 Department of Electrical Engineering and Information Technology, Vietnamese-German University, Hồ Chí Minh, Vietnam

2 Faculty of Engineering, Bristol Robotics Laboratory, Bristol University, Bristol, United Kingdom

References

[1] Lavalle SM, Kuffner JJ Jr. Rapidly-exploring random trees: Progress and prospects. In: Algorithmic and Computational Robotics: New Directions. 2000. pp. 293-308. DOI: 10.1.1.38.1387

[2] N. Preda, A. Manurung, O. Lambercy, R. Gassert and M. Bonfè. Motion planning for a multi-arm surgical robot using both sampling-based algorithms and motion primitives. In: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS); Hamburg. 2015. pp. 1422-1427. DOI: 10.1109/IROS.2015.7353554

[3] Ferguson D, Stentz A. Field D*: An interpolation-based path planner and replanner. Journal of Robotics Research. 2007:239-253. DOI: 10.1007/978-3-540-48113-3_22

[4] Stentz A. Optimal and efficient path planning for partially-known environments. In: Proceedings of the 1994 IEEE International Conference on Robotics and Automation; San Diego, CA. 1994. pp. 3310-3317. DOI: 10.1109/ROBOT.1994.351061

[5] Matthies L, Xiong Y, Hogg R, Zhu D, Rankin A, Kennedy B, Hebert M, Maclaehlan R, Won C, Frost T, Sukhatme G, Mchenry M, Goldberg S. A portable, autonomous, urban reconnaissance robot. In: Proceedings of the International Conference on Intelligent Autonomous Systems; 2000. DOI: 10.1.1.98.9353

[6] Doan KN, Le AT, Le TD, Peter N. Swarm robots' communication and cooperation in motion planning. In: Zhang D, Wei B, editors. Mechatronics and Robotics Engineering for Advanced and Intelligent Manufacturing. Cham: Springer; 2016. pp. 191-205. DOI: 10.1007/978-3-319-33581-0_15

[7] Hwang YK, Ahuja N. Gross motion planning—A survey. ACM Computing Survey. 1992;**24**(3):219-291. DOI: 10.1145/136035.136037

[8] Dechter R, Pearl J. Generalized best-first search strategies and the optimality of a*. Journal of the ACM (JACM). 1985;**32**(3):505-536. DOI: 10.1145/3828.3830

[9] Li J, Liu S, Zhang B, Zhao X. RRT-A* motion planning algorithm for non-holonomic mobile robot. In: 2014 Proceedings of the SICE Annual Conference (SICE); Sapporo. 2014. pp. 1833-1838. DOI: 10.1109/SICE.2014.6935304

[10] Arismendi C, Álvarez D, Garrido S, Moreno L. Nonholonomic motion planning using the fast marching square method. International Journal of Advanced Robotic Systems. 2015;**12**:5. DOI: 10.5772/60129

[11] Sun X, Yeoh W, Koenig S. Moving target D* Lite. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '10); Toronto, Canada. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems; p. 67-74. ISBN: 978-0-9826571-1-9

[12] Likhachev M, Ferguson D, Gordon G, Stentz A, Thrun S. Anytime dynamic A*: An anytime, replanning algorithm. In: Biundo S, Myers KL, Rajan K, editors. Proceedings of the Fifteenth International Conference on International Conference on Automated Planning and Scheduling (ICAPS'05); Monterey, California, USA. AAAI Press; 2005. p. 262-271. ISBN:1-57735-220-3

[13] Aine S, Likhachev M. Truncated incremental search. Journal of Artificial Intelligence. 2016;**234**(C):49-77. DOI: 10.1016/j.artint.2016.01.009

[14] Aine S, Likhachev M. Anytime truncated D*: Anytime replanning with truncation. In: Sixth Annual Symposium on Combinatorial Search; AAAI Publications. 2013

[15] Nash A, Koenig S, Likhachev M. Incremental Phi*: Incremental any-angle path planning on grids. In: Kitano H, editor. Proceedings of the 21st International Joint Conference on Artificial intelligence (IJCAI'09); 2009; San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2009. p. 1824-1830

[16] Le AT, Bui MQ, Le TD, Peter N. D* Lite with reset: Improved version of D* Lite for complex environment. In: First IEEE International Conference on Robotic Computing (IRC); Taichung, Taiwan. IEEE; 2017. pp. 160-163. DOI: 10.1109/IRC.2017.52

[17] Miao H, Tian YC. Robot path planning in dynamic environments using a simulated annealing based approach. In: 2008 10th International Conference on Control, Automation, Robotics and Vision; Hanoi. 2008. pp. 1253-1258. DOI: 10.1109/ICARCV.2008.4795701

[18] Likhachev M, Gordon G, Thrun S. ARA*: Anytime A* with Provable Bounds on Sub-Optimality. In: Proceedings of the 2003 Conference Advances in Neural Information Processing Systems 16 (NIPS-03); MIT Press; 2004. DOI: 10.1.1.3.9449

[19] Sven Koenig and Maxim Likhachev. D*lite. In: Rina Dechter, Michael Kearns, and Rich Sutton, editors. In Eighteenth National Conference on Artificial Intelligence; Edmonton, Alberta, Canada. Menlo Park, CA, USA: American Association for Artificial Intelligence; 2002. p. 476-483. ISBN:0-262-51129-0

[20] Sven Koenig, Maxim Likhachev, David Furcy. Lifelong Planning A∗. Artificial Intelligence. 2004;**155**(1):93-146. DOI: http://dx.doi.org/10.1016/j.artint.2003.12.001

[21] Daniele Frigioni, Alberto Marchetti-Spaccamela, and Umberto Nanni. Fully dynamic algorithms for maintaining shortest paths trees. Journal of Algorithms. 2000;**34**(2):251-281. DOI: http://dx.doi.org/10.1006/jagm.1999.1048

[22] Sethian JA. A fast marching level set method for monotonically advancing fronts. Proceedings of the National Academy of Sciences. 1996;**93**(4):1591-1595

[23] Prof. Ron Alterovitz. Configuration Space Visualization of 2-D Robotic Manipulator [Internet]. Available from: https://www.cs.unc.edu/~jeffi/c-space/robot.xhtml [Accessed: 8/18/2017]

Chapter 5

Path Planning on Quadric Surfaces and Its Application

Chi-Chia Sun, Gene Eu Jan, Chaomin Lu and Kai-Chieh Yang

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.72573

Abstract

In this chapter, recent near-shortest path-planning algorithms with $O(n\log n)$ in the quadric plane based on the Delaunay triangulation, Ahuja-Dijkstra algorithm, and ridge points are reviewed. The shortest path planning in the general three-dimensional situation is an NP-hard problem. The optimal solution can be approached under the assumption that the number of Steiner points is infinite. The state-the-art method has at most 2.81% difference on the shortest path length, but the computation time is 4216 times faster. Compared to the other $O(n\log n)$ time near-shortest path approach (Kanai and Suzuki, KS's algorithm), the path length of the Delaunay triangulation method is 0.28% longer than the KS's algorithm with three Steiner points, but the computation is about 31.71 times faster. This, however, has only a few path length differences, which promises a good result, but the best computing time. Notably, these methods based on Delaunay triangulation concept are ideal for being extended to solve the path-planning problem on the Quadric surface or even the cruise missile mission planning and Mars rover.

Keywords: Delaunay triangulation, Dijkstra algorithm, ridge point, near-shortest path, mission planning, NP-hard

1. Introduction

In the Euclidean plane with obstacles, the shortest path problem is to find an optimal path between source and destination. Shortest path algorithms have already been applied to motion planning of robots and path planning of navigation. Furthermore, it can be applied to electronic design automation (EDA), biological cell transportation and operation research (OR) [1–3].

In [4–6], Jan et al. proposed two $O(n\log n)$ time path-planning algorithms to obtain the near-shortest path in the Euclidian and quadric planes, respectively. Compared to the other

approaches of reduced visibility graph, this fast method outperforms the rest of $O(n\log n)$ algorithms in the general two-dimensional situation, except the path length compared to the shortest $O(n^2)$ time shortest algorithm of visibility graph.

In the quadratic plane, a survey of the shortest path problem concerning a two or higher dimensional geometric object (e.g. a surface, a polyhedron, space, network) can be found in [7]. The shortest path problem in the general three-dimensional situation is non-deterministic polynomial-time hard (NP-hard) problem [8], and only exponential time algorithms are known. In [9], the shortest path on a polyhedron is its local, which has an important property called unfolding, where the path must enter and leave at the same angle to the intersecting edge.

It follows that any locally optimal shortest path joining two consecutive obstacle vertices can be unfolded at each edge along its edge sequence, thus obtaining a straight segment. Sharir and Schorr [10] proposed an $O(n^3\log n)$ algorithm, which first applied this property to find the exact shortest path on a convex surface, where n is the number of edges. Later, Mitchell et al. [11] proposed an $O(n^2\log n)$ algorithm for propagating the shortest path map over a surface by a continuous Dijkstra method for general polyhedron. Chen and Han [12] improved it to an $O(n^2)$ algorithm. Faster algorithms than these cannot be found by far.

Kimmel and Sethian [13] presented a fast searching method for solving the Eikonal equation on a rectangular orthogonal mesh in $O(M\log M)$ steps, where M is the total number of grid points. They extended the fast marching method to triangulated domains with the same computational complexity. As an application, they provide an optimal time algorithm for computing the geodesic distances and thereby extracting shortest paths on triangulated manifolds.

Helgason et al. [14] presented a heuristic algorithm based on geometric concepts for the problem of finding a path composed of line segments from a given destination in the presence of polygonal obstacles. The basic idea involves constructing circumscribing triangles around the obstacles to be avoided. Their heuristic algorithm considers paths composed primarily of line segments corresponding to partial edges of these circumscribing triangles and uses a simple branch-and-bound procedure to find a relatively short path of this type.

Kanai and Suzuki proposed a near-shortest path approach (Kanai and Suzuki, KS's algorithm [9]) based on the Delaunay triangulation, the Dijkstra algorithm, and Steiner points, with computational complexity of $O(k^2n\log k^2n)$, where k is the number of Steiner points and n is the number of the triangles. Although KS's algorithm is an approximation, it has the significant advantages of easy implementation, high approximation accuracy, and numerical robustness. However, to obtain a shorter path, the computation time required by the path planning will increase rapidly when the Steiner points increases. A detailed comparison can be found in **Table 1**.

In this chapter, an $O(n\log n)$ time near-shortest path planning is introduced. It combined with the Delaunay triangulation, Ahuja-Dijkstra algorithm, and ridge points for path planning on a quadratic surface. Experimental results show that the average path length of the Delaunay triangulation-based algorithm is 0.28% longer than the KS's algorithm; however, the speed is 31.71 times faster. Furthermore, when performing KS's algorithm with 29 Steiner points, the NP-hard shortest path will be found (extremely close approximation of the shortest path planning). Although the length is 2.81% longer than the shortest, the computation time is

Euclidean	Algorithms				
	Polyhedron			Polyhedron or quadric	
	Sharir [10]	Mitchell [11]	Chen [12]	KS's [9]	Delaunay method
Connection	No	No	No	$6k^2n$	$9n$
Time complexity	$O(n^3\log n)$	$O(n^2\log n)$	$O(n^2)$	$O(k^n\log k^2n)$	$O(n\log n)$
Is the path shortest?	N/A	N/A	N/A	Near-shortest	Near-shortest

Table 1. Comparison of different shortest path algorithms in the three-dimensional space, ***n*** denotes the number of triangle mesh.

4216 times faster. Therefore, it can not only obtain a good near-shortest path length on the quadratic surface, but also improve the computation time. Furthermore, it is worth noting that Delaunay triangulation-based fast algorithms are ideal for being extended to solve the path planning in the polyhedron plane or be applied to cruise missile mission planning in the quadratic plane.

This chapter is organised as follows. Section II briefly introduces the concept of shortest path algorithms. In Section III, we will describe the idea of the triangulation-based near-shortest path algorithm, the performance of which is analysed in Section IV. The experimental results are shown in Section V. Section VI explains a possible application for cruise missile mission planning, and Section VII concludes the chapter.

2. Algorithm backgrounds

In this section, basic concepts of the Delaunay triangulation algorithm on the quadratic surface will be introduced, such as Euclidean plane, Delaunay triangulation, ridge points, and Dijkstra's single-source shortest path algorithm. Euclidean space is the Euclidean plane and three-dimensional space of Euclidean geometry [15], as well as the generalisations of these notions to higher dimensions. It can be used to distinguish these spaces from the curved spaces of non-Euclidean geometry and Einstein's general theory of relativity [16].

For the quadratic surface, there is essentially only one Euclidean space with three real number coordinates from the modern viewpoint.

A Quadric surface is the locus of the points (x, y, z), which satisfy a second-degree equation in three variables, $Ax^2 + By^2 + Cz^2 + 2Dxy + 2Exy + 2Fyz + Gx + Hy + Iz + J = 0$, and the different types of Quadric surfaces, a total of 15, are obtained by varying the coefficients of it [17].

The classification of the different types of Quadric surfaces is made, first, on the basis of the matrix of the quadratic from determining by the symmetric matrix:

$$A_Q = \begin{pmatrix} A & D & E \\ D & B & F \\ E & F & C \end{pmatrix} \tag{1}$$

A triangle mesh is a type of polygon mesh in computer graphics. It comprises a set of triangles (typically in three dimensions) that are connected by their common edges or corners [18]. With individual triangles, the system has to operate on three vertices for every triangle. In mathematics and computational geometry, the triangle mesh can be expressed in a Delaunay triangulation for a set V of vertices in the plane is a triangulation DT(V) such that no vertex in V is inside the circle of any triangle in DT(V) [19].

A path that interconnects the two vertices v and v' of a graph G with minimal length for all paths is called the shortest path. Finding a shortest path in a graph G can be done in $O(n\log n)$ with Ahuja-Dijkstra's single-source shortest path algorithm by using Fibonacci heaps (F-heaps) and radix heaps [20].

A ridge is a curve consisting of ridge point: A point lies on a ridge if its neighbourhood can be subdivided by a line passing through it, and such that the surface in each half-neighbourhood is monotonically decreasing when moving away from the line [21].

3. Algorithm and illustration

In this section, a Delaunay triangulation-based method will combine the concepts of Ahuja-Dijkstra algorithm and ridge points to construct a directed graph and to obtain the shortest possible path length on the quadratic surfaces. Compared to another Delaunay triangulation method [4], Fermat points are replaced by the ridge points; this is mainly due to the fact that Fermat points cannot connect the shortest line between two neighbour triangles on the quadratic surface. The initial step of the algorithm is to build a triangle mesh G to simulate the earth's surface (the GIS map). Next, a source point and a destination point are spotted, then three ridge points in the same triangle will be connected together to generate an extra small triangle and three extra path segments between the vertices and neighbour triangles diagonal vertices.

These extra connections as shown in **Figure 1(b)** will be used to search for the near-shortest path by using the Ahuja-Dijkstras algorithm (expressed by E_t). As we have constructed the directed connected graph $G' = G \cup E_t$, we can obtain the near-shortest path P in graph G' if it exists.

Function 1: *FindingExtraConections*():

Step 1. Create a ridge point between two neighbouring triangles as shown in **Figure 1(a)**.

Step 2. Obtain the shortest path by connecting these two vertices E, F with the ridge point.

END {Function of *FindingExtraConections*}.

Function 2: *PathShortening*(P):

Step 1. Generate the shortcuts of any two consecutive segments by performing the Function 1.

Step 2. Sort the shortcuts by their corresponding length improvements in a descending order.

Step 3. Shorten the original path in a descending order.

END {Function of *PathShortening*}.

Algorithm: The triangle mesh-based shortest path on the quadric surfaces.

Init Load the data from a GIS map.

Step 1. Construct a triangle mesh G by the Delaunay triangulation on the data, locate a source point and a destination point.

Step 2. Compute the shortest path between the neighbouring vertices based on the ridge points on the quadric surface by performing Function 1.

Step 3. Construct the directed connected graph G' with the extra connections.

Step 4. Obtain the shortest path in graph G' if it exists by Ahuja-Dijkstras algorithm.

Step 5. Call the Function 2 *PathShortening*(P).

END {Algorithm of the triangle mesh-based shortest path}.

Figure 2 illustrates the detailed process. **Figure 2(a)** shows the initiation of a triangle mesh G, then a source point S and a destination point D are depicted in **Figure 2(b)**. Next, ridge points will be inserted into the triangle mesh in order to generate extra path connections

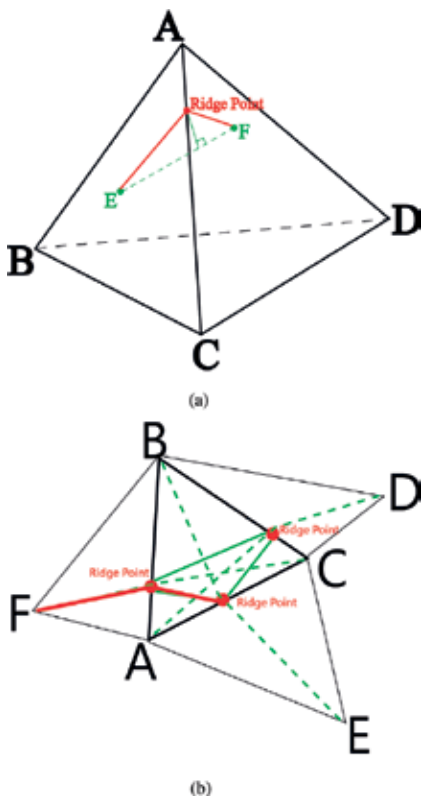

Figure 1. Illustration of the ridge points. (a) A ridge point of ΔABC and ΔACD. (b) The connections between the ΔABC and neighbour triangles vertices and ridge points.

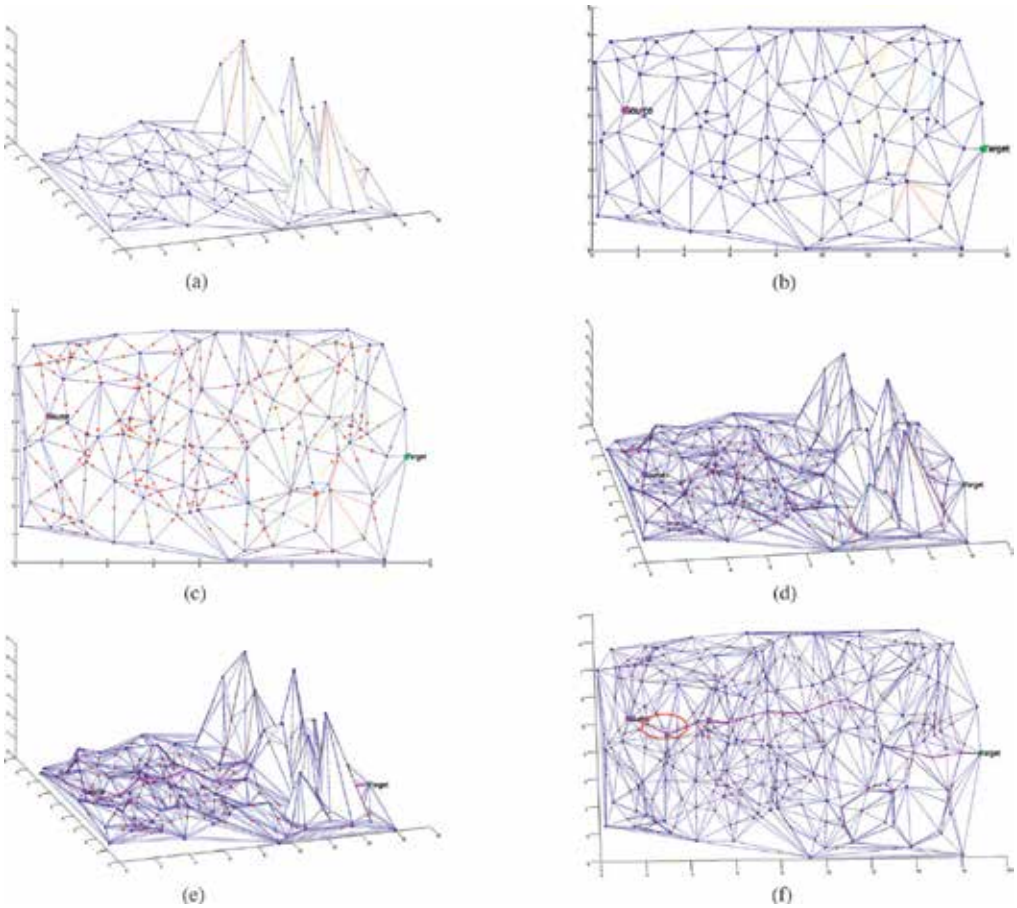

Figure 2. Illustration of the Delaunay triangulation algorithm. (a) Initialise (aerial view). (b) Spot source point and destination point (top view). (c) Insert ridge points (top view). (d) Extra connections (aerial view). (e) Obtain the shortest path (aerial view). (f) PathShortening (top view).

as shown in **Figure 2(c)**. **Figure 2(d)** shows that each ridge point will connect with the others in the same triangle and three extra path segments between the vertices and neighbour triangles diagonal vertices. Thereafter, the shortest path can be obtained in the graph G' by Ahuja-Dijkstras algorithm presented as a red line shown in **Figure 2(e)**. Finally, **Figure 2(f)** shows the final result after the *PathShortening*.

4. Performance analysis

A near-shortest path algorithm on the Quadratic surface is the fastest in the literature.

Theorem 1. The time complexity of the algorithm in the triangle mesh G is $O(n\log n)$, where n denotes the number of triangles.

Proof: We can generate Delaunay triangulation as a triangle mesh with time of $O(n\log n)$ [22], as a result, time complexity in Step 1 is $O(n\log n)$ [23].

The number of the ridge points is bounded by $O(n)$; thus, the number of connections in Step 3 is also bounded by $9 \times O(n)$. We therefore know that all the time complexity for Steps 2 and 3 are $O(n)$. In Step 4, the time complexity of the Ahuja-Dijkatras algorithm using Fibonacci heaps and radix heaps is $O(n + n\log n) = O(n\log n)$ [20]. In Step 5, as the number of points are $n - 3$, the time complexity will be dominated by $O[(n - 3)log(n - 3)] = O(n \log n)$. Therefore, the overall time complexity for *PathShortening* is $O(n\log n)$.

In conclusion, the time complexity is $9 \times O(n) + O(n \log n) = O(n \log n)$ from Steps 1–5.

Theorem 2. The space complexity of constructing the triangle mesh in the quadratic surface is $O(n)$, where n is the number of triangles.

Proof: According to Euler characteristic, the number of triangles is less than $T = 2 \times (k_C \times n) - h - 2$ once the triangle mesh is generated, where h is the number of corners of the triangles. Therefore, the space complexity is bounded by $O(n)$. Furthermore, the number of edges including the original edges of triangles, edges connecting ridge points to the vertices of triangles, and connection between ridge points is also bounded by $(6 + 3/2) \times T = 7.5\ T$. Hence, the space complexity is $O(n)$.

5. Experimental result

The performance of Delaunay triangulation-based path algorithm has been analysed for evaluating the near-shortest path with several real GIS maps in the Matlab Language. The analysis was performed on an Intel Core2 Quad CPU Q9550@2.83 GHz processor with 8 GB memory. **Figure 3** shows one of the experimental results with a GIS map, where the solid line is the near-shortest path and dashed lines are the shortcuts.

Next, we have compared this algorithm to the KS's algorithm with 1, 3, 5, 7, 9, 19 and 29 Steiner points and summarised the comparison results on the average path length and the average runtime in **Table 2**. In KS's algorithm, each edge of the triangle has been divided into multiple segments to generate more connections for path searching. **Figure 4(a)** and **(b)** illustrates the average running time and path length between two algorithms.

When compared to one Steiner point, the average path length difference of the Delaunay triangulation-based algorithm is 6.14% better than the KS's algorithm, and computation time between the Delaunay triangulation-based algorithm and the KS's algorithm is same. When it increased three Steiner points, the length difference is only 0.28%, but the computation time is 31.71 times faster. When 29 Steiner points for the KS's algorithm are applied, the KS's results can be assumed as the shortest path; however, the length difference is 2.81% longer and computation time is 4216 times faster. This proves that the Delaunay triangulation-based algorithm can solve the NP-hard problem and also obtain fast computing features. From the statistical view, **Figure 5** shows the prediction of the average computation time and length difference if the number of KS's Steiner points is infinity.

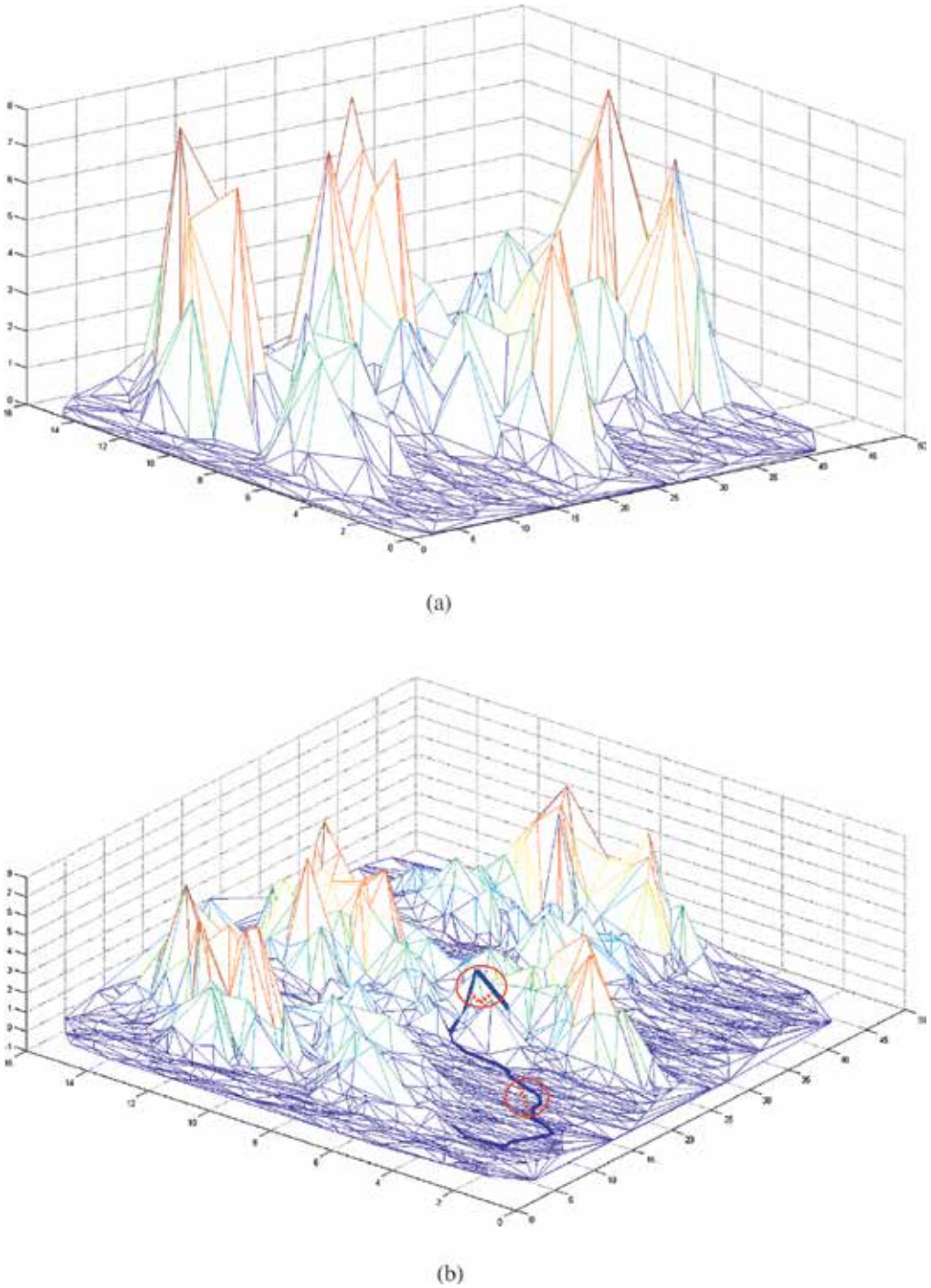

Figure 3. Near-shortest path searching with a GIS map. (a) Initialise (aerial view) and (b) result (aerial view).

SPs	Length difference (%)	Runtime difference (X) (%)
1	−6.14	0.97
3	0.28	31.71
5	1.66	86.40
7	2.26	162.94
9	2.68	414.55
19	2.79	1968.62
29	2.81	4215.75
999 ≅ ∞	5.3	3.0E + 10

Table 2. Comparisons between our algorithm and KS's algorithm on average running time and length difference when the Steiner points are 1, 3, 5, 7, 9, 19, 29, …, ∞

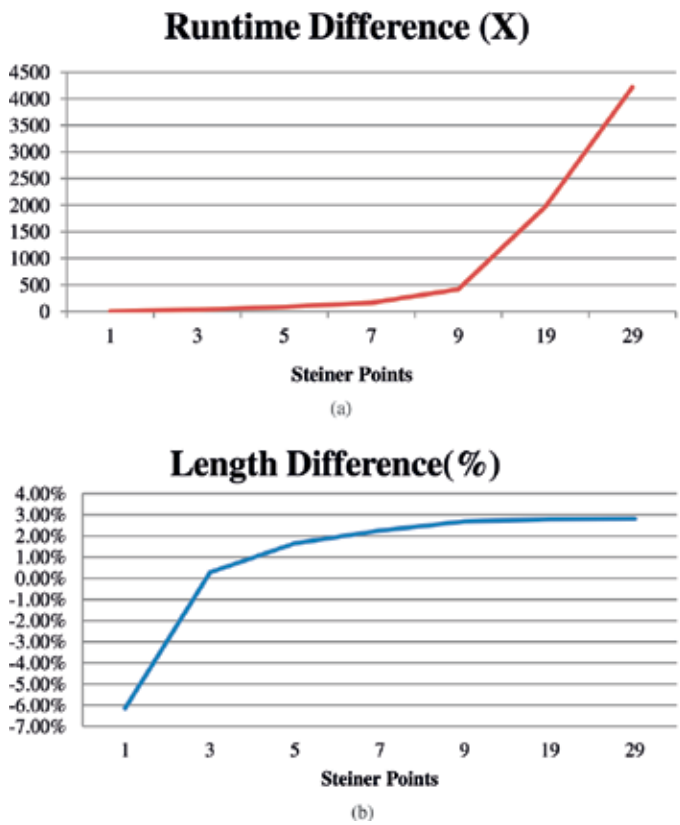

Figure 4. Comparison between Delaunay triangulation-based algorithm and KS's algorithm on average running time and path length. (a) Average computation time and (b) average length difference.

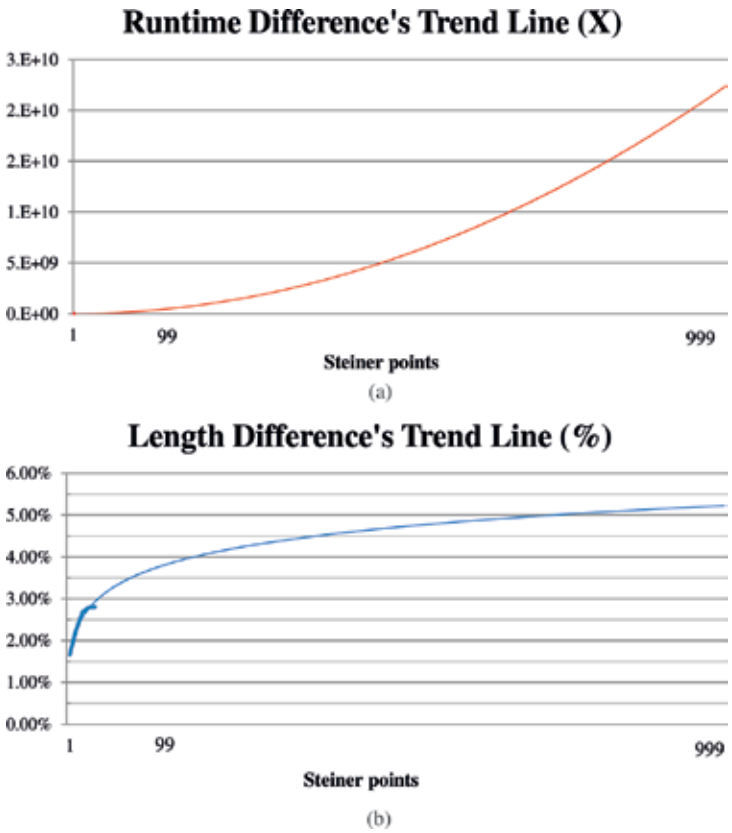

Figure 5. The prediction of the average computation time and length difference if the number of KS's Steiner points is infinity. (a) Average computation time and (b) average length difference.

6. The shortest path application on the quadric surface

This section explains an application that benefits from the Delaunay triangulation-based algorithm. Actually, it can be applied to shortest path planning for Mars rover and mission planning for cruise missiles in the quadric surface. For cruise missile mission planning, we steady up with the angle θ_1 at source point and down with the angle θ_2 at destination point, respectively. Furthermore, the z-coordinate is limited by the l altitude units to avoid the radar's scan as well as crash prevention, where l is a constant, as shown in **Figure 6(a)**. To verify the correctness and performance, we assume a cruise missile needs to move from the source position S to the destination position D, as shown in **Figure 6(b)**. In order to keep the safety margin between the cruise missile and quadric surface, virtual l altitude units are added up to the graph G' (e.g. 20 meters above the G). Once the virtual altitude and thresholds are applied, a shortest path is obtained. Apparently, **Figure 6** shows that this shortest path algorithm can be also applied to intelligently guide the cruise missile to pass a narrow passage and avoid radar's scan.

http://dx.doi.org/10.5772/intechopen.72573

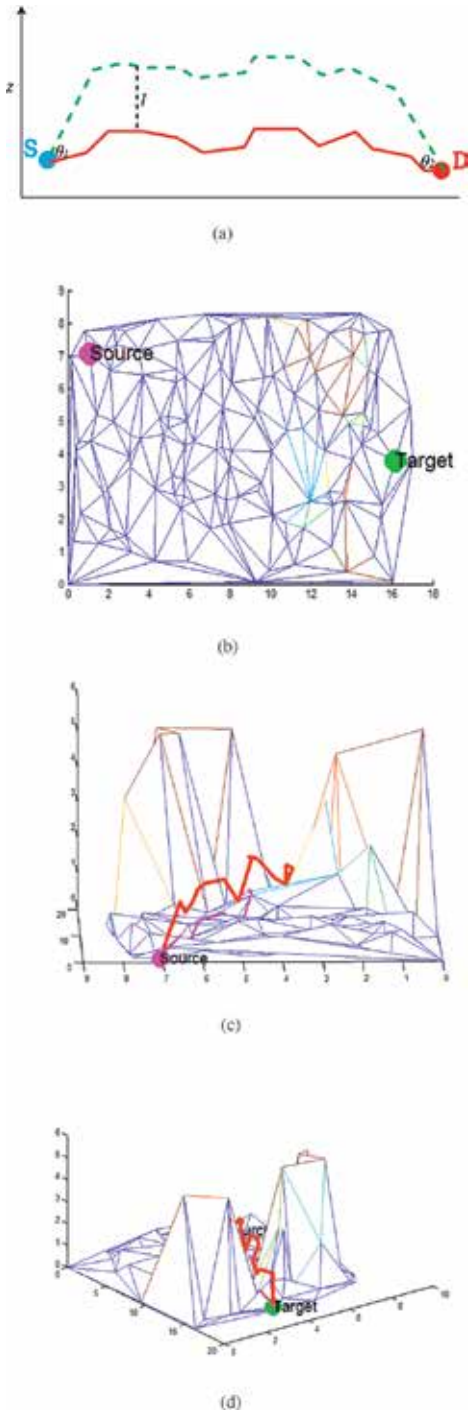

Figure 6. An illustration of the shortest path for planning a cruise missile on the landscape. (a) Cruise missile planning; (b) land scope (top view); (c) result (aerial view 1) and (d) result (aerial view 2).

7. Conclusion

In this chapter, an $O(n\log n)$ time near-shortest path planning based on the Delaunay triangulation, the Ahuja-Dijkstra algorithm, and ridge points on the quadric surface are introduced. Although the length of path obtained by Delaunay triangulation-based algorithm is 0.28% longer than another $O(n\log n)$ time KS's algorithm, the average computation time is 31.71 times faster. Furthermore, when the KS's Steiner point is 29, which means that the shortest path in the NP-hard problem will be obtained, the Delaunay triangulation-based algorithm has at most a 2.81% difference on the path searching, but the computation time is 4216 times faster approximately. Therefore, the Delaunay triangulation-based algorithm presents a good near-shortest path searching solution in the quadric surface with a very short amount of computation time.

Author details

Chi-Chia Sun[1], Gene Eu Jan[2]*, Chaomin Lu[3] and Kai-Chieh Yang[4]

*Address all correspondence to: geneeujan@gmail.com

1 Department of Electrical Engineering, National Formosa University, Taiwan, ROC

2 Tainan National University of the Arts, Taiwan, ROC

3 Department of Electrical and Computer Engineering, University of Detroit Mercy, USA

4 Department of Electrical Engineering, National Taiwan Ocean University, Taiwan, ROC

References

[1] Wu Y, Sun D, Huang W, Xi N. Dynamics analysis and motion planning for automated cell transportation with optical tweezers. IEEE/ASME Transactions on Mechatronics. 2012;**18**(2):706-713. DOI: http://dx.doi.org/10.1109/TMECH.2011. 2181856

[2] Harada K, Hattori S, Hirukawa H, Morisawa M, Kajita S, Yoshida E. Two-stage time-parametrized gait planning for humanoid robots. IEEE/ASME Transactions on Mechatronics. Oct 2010;**15**(5):694-703. DOI: http://dx.doi.org/10.1109/TMECH.2009.2032180

[3] Jan GE, Chang KY, Parberry I. Optimal path planning for mobile robot navigation. IEEE/ASME Transactions on Mechatronics. Aug 2008;**13**(4):451-460. DOI: http://dx.doi.org/10.1109/TMECH.2008.2000822

[4] Jan GE, Sun CC, Tsai WC, Lin TH. An $O(n\log n)$ shortest path algorithm based on Delaunay triangulation. IEEE/ASME Transactions on Mechatronics. Apr 2014;**19**(2):660-666. DOI: http://dx.doi.org/10.1109/TMECH.2013.2252076

[5] Sun CC, Jan GE, Leu SW, Yang KC, Chen YC. Near-shortest path planning on a quadratic surface with $O(n\log n)$ time. IEEE Sensors Journal. Nov 2015;**15**(11):6079-6080. DOI: http://dx.doi.org/10.1109/JSEN.2015.2464271

[6] Jan GE, Fung K, Wu PY, Leu SW. Shortest path-planning on polygonal surfaces with $O(n\log n)$ time. In: IEEE International Conference on Control and Robotics Engineering. IEEEXplore. Apr 2016. pp. 1-5. DOI: http://dx.doi.org/10.1109/ICCRE.2016. 7476149

[7] Mitchell JSB. The Geometric Shortest Paths and Network Optimization in the Handbook of Computational Geometry. North Holland: Elsevier Science; 1998

[8] Canny J, Reif J. New lower bound techniques for robot motion planning problems. In: Annual Symposium on Foundations of Computer Science. IEEEXplore. 1987. pp. 49-60. DOI: http://dx.doi.org/10.1109/SFCS.1987.42

[9] Kanaia T, Suzuki H. Approximate shortest path on a polyhedral surface and its applications. Computer-Aided Design. Sep 2001;**33**(11):801-811. DOI: https://doi.org/10.1016/S0010-4485(01)00097-5

[10] Sharir M, Schorr A. On shortest paths in polyhedral spaces. SIAM Journal of Computing. 1986;**15**:193-215. DOI: http://dx.doi.org/10.1137/0215014

[11] Mitchell JSB, Mount DM, Papadimitriou CH. The discrete geodesic problem. SIAM Journal on Computing. 1987;**16**(4):647-668. DOI: https://doi.org/10.1137/0216045

[12] Chen J, Han Y. Shortest paths on a polyhedron. In: ACM Symposium on Computational Geometry. ACM Digital Library. 1990. pp. 360-369. DOI: http://dx.doi.org/10.1145/98524.98601

[13] Kimmel R, Sethian JA. Computing geodesic paths on manifolds. In: Proceedings of the National Academy of Sciences on Applied Mathematics. PNAS Online. July 1998;**95**: 8431-8435

[14] Helgason R, Kennington J, Lewis K. Cruise missile mission planning: A heuristic algorithm for automatic path generation. Journal of Heuristics. Sep 2001;**7**(5):473-494. DOI: http://dx.doi.org/10.1023/A:1011325912346

[15] Byer O, Lazebnik F, Smeltzer DL. Methods for Euclidean Geometry. Washington D.C. USA: Mathematical Asso-ciation of America; 2010

[16] Einstein A. Die Feldgleichungen der Gravitation. Sitzungsberichte der Preussischen Akademie der Wissenschaften zu Berlin. 1915;**48**:844-847. DOI: http://dx.doi.org/10.1002/3527608958.ch5

[17] Galarza R, Irene A, Seade J. Introduction to Classical Geometries. Berlin, Germany: Springer; 2007. DOI: http://dx.doi.org/10.1007/ 978-3-7643-7518-8

[18] Jin J. Three Novel Algorithms for Triangle Mesh Processing: Progressive Delaunay Refinement Mesh Generation, MLS-based Scattered Data Interpolation and Constrained Centroid Voronoi-based Quadrangulation. IL, USA: UMI Dissertation Publishing; 2011

[19] Cheng S-W, Dey TK, Shewchuk J. Delaunay Mesh Generation. UK: Chapman and Hall/CRC; 2012

[20] Ahuja R, Mehlhorn K, Orlin J, Tarjan R. Faster algorithms for the shortest path problem. Journal of the ACM. Apr 1990;**37**:213-223. DOI: http://dx.doi.org/10.1145/77600.77615

[21] Sack J, Urrutia J. Handbook of Computational Geometry. North Holland: Elsevier; 1999

[22] Fortune S. A sweepline algorithm for voronoi diagrams. In: Proceedings of the Second Annual ACM Symposium on Computational Geometry. Berlin, Germany: Springer-Verlag; 1986. pp. 313-322. DOI: https://doi.org/10.1007/BF01840357

[23] Rohnert H. Shortest paths in the plane with convex polygonal obstacles. Information Processing Letters. 1986:**23**(2):71-76. DOI: http://dx.doi.org/10.1016/0020-0190(86)90045-1

Section 2

Extended Path Planning for Mobile Robots

Chapter 6

Path Planning in Rough Terrain Using Neural Network Memory

Nancy Arana-Daniel, Roberto Valencia-Murillo,
Alma Y. Alanís, Carlos Villaseñor and
Carlos López-Franco

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.71486

Abstract

Learning navigation policies in an unstructured terrain is a complex task. The Learning to Search (LEARCH) algorithm constructs cost functions that map environmental features to a certain cost for traversing a patch of terrain. These features are abstractions of the environment, in which trees, vegetation, slopes, water and rocks can be found, and the traversal costs are scalar values that represent the difficulty for a robot to cross given the patches of terrain. However, LEARCH tends to forget knowledge after new policies are learned. The study demonstrates that reinforcement learning and long-short-term memory (LSTM) neural networks can be used to provide a memory for LEARCH. Further, they allow the navigation agent to recognize hidden states of the state space it navigates. This new approach allows the knowledge learned in the previous training to be used to navigate new environments and, also, for retraining. Herein, navigation episodes are designed to confirm the memory, learning policy and hidden-state recognition capabilities, acquired by the navigation agent through the use of LSTM.

Keywords: robot navigation, learning to search, reinforcement learning, LSTM unstructured terrain, rough terrain, cost function

1. Introduction

Autonomous robot navigation in unstructured terrain allows a robot to move through an environment for which the selection of traversable terrain is not a deterministic decision [1]. A mobile robot must make decisions of when to traverse patches of terrain that could be dangerous or that might consume too many resources.

Several approaches have been developed in order to solve this problem; some of them are focused on classifying traversable terrain [2, 3], and others in coupling the perceptual and

planning systems of the robot using functions that map features of the environment to scalar values that represent the traversability of the terrain [1, 4, 5]. These works try to resolve the task of autonomous robot navigation in unstructured terrain; however, these approaches do not address the issue as an integrated system.

In most cases, a human expert provides information for cost map constructions heuristically. In other cases, this information is then used to construct a cost function that automatically maps features to costs; however, note that this cost function is established heuristically. The problem with this methodology is that the traversability of a given feature for a robot is difficult to quantify. In contrast, humans can determine traversal trajectories relatively easily.

An alternative to establishing cost functions is to use an algorithm that automatically constructs and tunes a cost function. Learning to Search (LEARCH) [1] is an algorithm that uses learning from demonstration in order to construct a cost function. In this approach, a human expert exhibits a desirable behavior (a sample path) over certain terrain; then, LEARCH adjusts a cost function in order to match the behavior exhibited by the expert.

LEARCH has the advantage that the cost function to be constructed can be chosen from among linear functions, parametric functions, neural networks and decision trees, to name a few [1]. In particular, neural networks and other learning machines such as support vector machines (SVM) have exhibited considerable generalization capability in different scenarios [6].

However, as will be demonstrated in this paper, the LEARCH generalization capability decreases as the number of sample paths increases; this is because the error decays over time during training.

In this study, this problem with the LEARCH algorithm is addressed using a long-short-term memory (LSTM) neural network and reinforcement learning (RL). Recurrent neural networks are capable of finding hidden states, as shown in [7]. Therefore, we propose a complex learning system that allows a navigation agent to learn navigation policies and determine complex traversability cost functions. Furthermore, this system can retain the knowledge learned in past navigation episodes in memory and generalize this knowledge for use in new episodes.

2. Learning to Search

This section presents an overview of the LEARCH algorithm, along with the results of generalization tests conducted using LEARCH. The objective is to explain the need for memory for this algorithm in order to improve its performance.

The LEARCH algorithm is based on the concept of inverse optimal control [8], which addresses the problem of finding a cost map such that a known trajectory through an environment is optimally navigated using this map. In addition, non-linear maximum margin planning [1] with the support vector regression machine [9] is used to learn behavior from an expert, that is, human expert.

Let S be a state space operated by a path planner. F is a feature space defined over S. Then, for every $x \in S$ a corresponding feature vector $F_x \in S$ exists. The F_x vectors are inputs for the cost

function C, which maps F to scalar values. C is defined as the weighted sums of functions $R_i \in \mathcal{R}$, where $\mathcal{R}$ is a space of limited complexity that maps from the feature space to a scalar [1].

We define a path P as a sequence of states $x \in S$ that lead from the start s point to the goal g. The cost of each state is $C(F_x)$; thus, the cost of the entire path is defined as

$$C(P) = \sum_{x \in P} C(F_x) \tag{1}$$

Consider a path provided by an expert, i.e. sample path P_e, which runs from a start state s_e to a goal state g_e. In order to learn from the expert demonstration, a cost function such that P_e is the optimal path from s_e to g_e is required. This task can be expressed as the following optimization problem [1]:

$$\text{Min}O[C] = \lambda \text{REG}(C) + \sum_{x \in P} C(F_x) - \min \widehat{P} \left[\sum_{x \in \widehat{P}} (C(F_x) - L_e(x)) \right] \tag{2}$$

where λ is a term that scales the regularization term REG(C). $\widehat{P}$ is a path computed by a planner over the cost space, and L_e is a loss function that encodes the similarity between paths. The latter is defined as

$$L_e = \begin{cases} 1 \text{ if } x \in P_e, \\ 0 \text{ otherwise} \end{cases} \tag{3}$$

The sub-gradient is used to minimize $O[C]$. In the cost function space, the sub-gradient is

$$\nabla O_F[C] = \lambda \nabla REG_F[C] + \sum_{x \in P_e} \delta_F(F_x) - \sum_{x \in P_*} \delta_F(F_x), \tag{4}$$

where δ is the Dirac delta and P_* is the optimal path for the actual cost map.

In order to avoid overfitting, a cost function space is considered. C is now defined as the space of weighted sums of functions $R_i \in \mathcal{R}$, where $\mathcal{R}$ is a space of functions of limited complexity, which maps from the feature space to a scalar. The possible choices for $\mathcal{R}$ include linear functions, parametric functions, neural networks and decision trees. Thus,

$$\begin{gathered} C = \left\{ C | C = \sum_i \eta_i R_i(F), R_i \in \mathcal{R}, \eta_i \in \mathbb{R} \right\} \\ \mathcal{R} = \{ R | R : \mathcal{F} \to \mathbb{R} \wedge REG(R) < v \} \end{gathered} \tag{5}$$

The functional gradient is projected onto the direction set by finding the element $R_i \in \mathcal{R}$ that maximizes the inner product $\langle -\nabla O_F[C], R_* \rangle$. This maximization can be regarded as a learning problem. Here,

$$\begin{aligned} R_* &= \arg\max_R \langle -\nabla O_F[C], R_* \rangle \\ &= \arg\max_R \sum_{x \in P_e \cap P_*} \alpha_x y_x R(F_x) \end{aligned} \tag{6}$$

where

$$\alpha_x = |\nabla O_{F_x}[C]| y_x = -\operatorname{sgn}(\nabla O_{F_x}[C])$$

As in [1] the projection of the functional gradient can be regarded as a weighted classification problem. It can be seen that the regression targets y_x are positive in regions of the feature space for which the planned path visits more than the sample path and negative in the opposite case. Here, this approach is viewed as minimizing the error induced by visiting states that are not in the sample path. Then, the visitation count U is the cumulative count of the number of states $x \in P$ such that $F_x = F$. The visitation counts can be split into positive and negative components, depending on whether they correspond to the current planned path or the sample path:

$$\begin{aligned} U_+(F) &= \sum_{x \in P_*} \delta_F(F_x) \\ U_-(F) &= \sum_{x \in P_e} \delta_F(F_x) \end{aligned} \tag{7}$$

$$U(F) = U_+(F) - U_-(F) = \sum_{x \in P_*} \delta_F(F_x) - \sum_{x \in P_e} \delta_F(F_x) \tag{8}$$

Ignoring the regularization term of Eq. (4), the regression targets and weights can be computed as functions of the visitation counts. Then, the regressor targets can be obtained using these visitation counts. Further, with this regressor, the cost function can be expressed as

$$C_j = C_{j-1} * e^{n_j R_j} \tag{9}$$

where $j = \{1, 2, 3, \ldots, n\}$, with n being the number of iterations; R is the regressor; and η is the learning rate.

2.1. Learning to Search experiments

This section describes the experiment conducted to test the LEARCH generalization capabilities. Satellite-like images were selected for feature extraction. Further, patches of terrain were divided into grid cells, with a vector being created for each cell. These vectors represented a value for each of the following features of the environment in each dimension: the vegetation density, slope, the presence of gravel or rocks and the presence of water; each scalar value represents an abstraction of the feature; for example, the scalar value for vegetation represents its density, a patch of terrain with grass would be represented with a low value, and a patch of terrain with a tree would be represented with a high value of vegetation. In these experiments vectors of dimension 4 were used, that is, for the patch of terrain with grass, the vector [0, 2, 0, 0] would be its representation. A human expert-traced sample paths over the terrain, as shown in **Figure 1**.

Figure 1. Left: Satellite like image. Right: Grid cells and lines with different colours representing sample paths.

With this information a support vector regressor (SVR) was used to learn the cost function R_i of Eq. (5). Note that, after LEARCH is executed, the trained SVR can map the features of the terrain directly into traversal costs. **Figure 2** shows a diagram of the algorithm used for training.

The procedure is explained as follows:

- A cost map M is constructed using a feature map and a cost function ($C(F)$).
- The path planner D^* computes an optimal path P^* from start point s_e to the goal point g_e over M.
- Using the sample path from the expert path P_e and P^*, the vector U indicating the visitation counts is constructed as shown in Eq. (8).
- Using U, the regressor targets are computed, and this regressor is trained.
- The cost function is updated using Eq. (9).
- This process is repeated until P_e and P^* are equal.

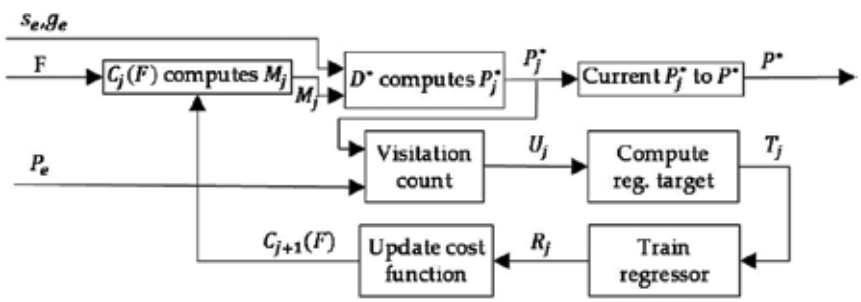

Figure 2. LEARCH diagram. F is the feature map, M is the cost map, s_e and g_e represent the start and the goal points, respectively, P^* is the optimal path, P_e is the sample path, T is the vector of regressor targets values for training a regressor R, C (F is the cost function and $j = \{1,2,3,\ldots, n\}$, where n is the number of iterations needed to train the cost function.

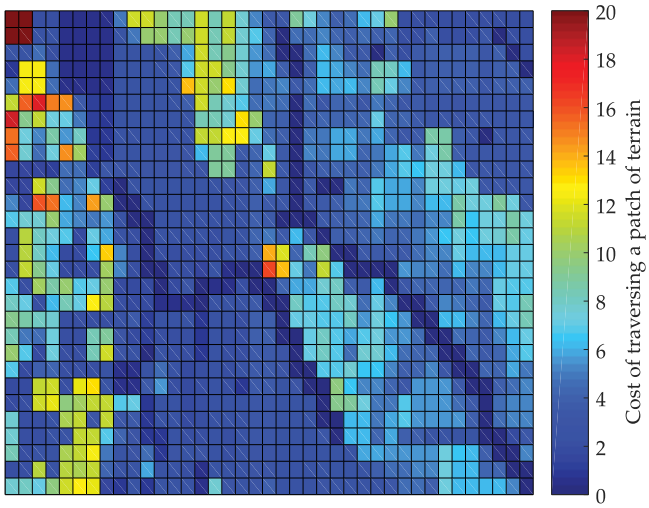

Figure 3. Example of a cost map which belongs to the real map shown in **Figure 1**.

The traversal costs can be color coded for demonstration purposes, as shown in **Figure 3**, where values near 20 represent the patches with the highest crossing difficulty and those near zero represent terrain that is easy to traverse.

In this experiment, in order to prove the generalization capabilities of LEARCH (i.e. its ability to use policies learned in past navigation episodes during new episodes), we employed the following methodology. First, we trained the LEARCH system described in **Figure 1** using an initial map and one sample path. Then, we incrementally added to this learned knowledge (using the same initial map) by incorporating more paths to be learned by LEARCH (one by one). The results of these experiments are shown in **Figure 4**, where the image (a) of the figure shows the cost map obtained with a cost function trained using one sample path and the image (b) shows the results obtained by adding a path to the training, and so on until five sample paths are used. The image (f) of **Figure 4** shows the cost map obtained with a cost function trained using eight sample paths.

From the cost map (e) of **Figure 4**, in comparison with the cost map (f), it is apparent that information from the environment is missing after the cost function is trained with more sample paths. That is, some states are no longer recognized as states with a high traversal cost. It is important to note that the costs that are most affected are those furthest from the sample paths, in comparison with the costs of the corresponding states on the original path; therefore, the generalization capability of the LEARCH system is very poor. This problem renders the task of finding the optimal path difficult. In addition, the path planner could compute a path that traverses dangerous terrain. Further, note that, the use of only a few sample paths is not a solution to the problem of obtaining a system with knowledge of a greater number of area costs than those attached to the sample paths. This is because such sample paths cannot contain all the information necessary for a good and complete representation of the environment.

http://dx.doi.org/10.5772/intechopen.71486

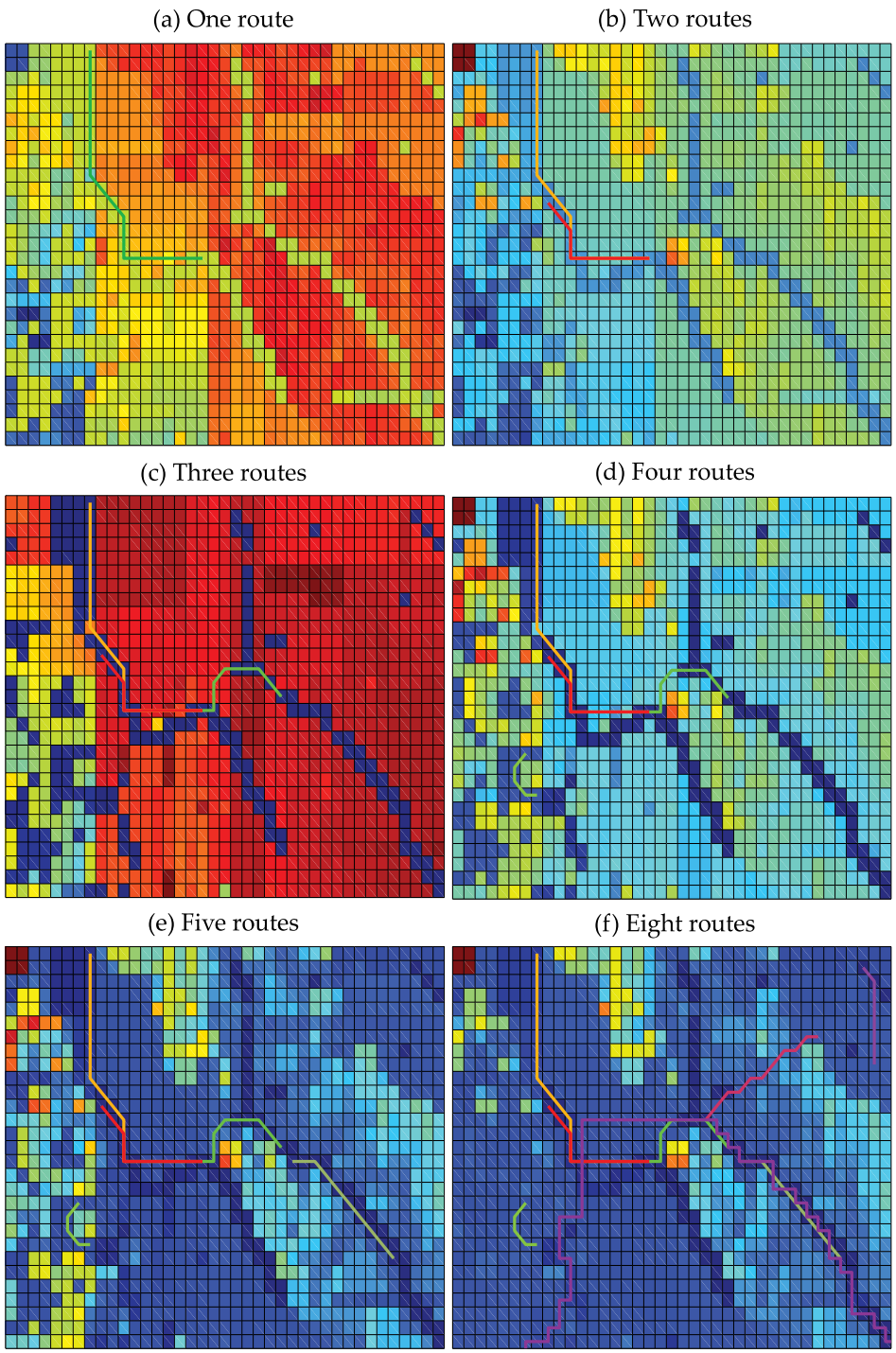

Figure 4. Costs maps obtained using different numbers of paths of terrain.

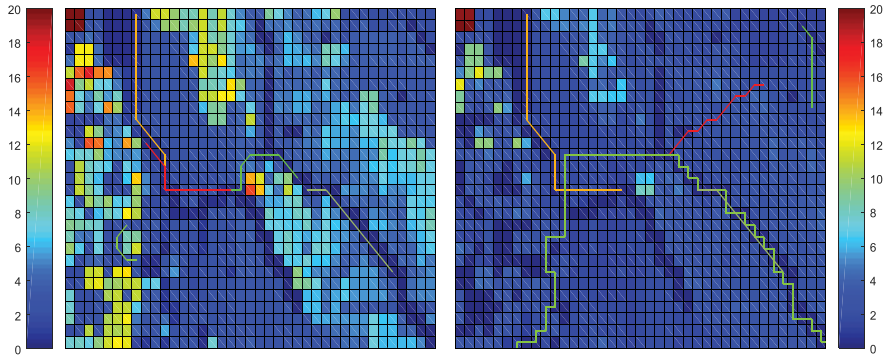

Figure 5. Left: cost map computed with five representative paths. Right: cost map computed with five non representative paths.

In this study, other experiments to prove the limitations of LEARCH were performed, in which we trained the system using nonrepresentative environment paths. That is, the paths taught by the expert traversed many cells of the environment that did not contain sufficient representative features of the environment or cells that did not have significant differences in cost. **Figure 5** shows examples of these paths, which allowed the LEARCH system to acquire nonrepresentative knowledge that was then generalized over the cost map. The cost map at the left of **Figure 5** is less generalized compared with the more descriptive costs shown on the map at the right of **Figure 5**.

Therefore, in order to address the problems with the LEARCH system, we propose the use of an LSTM as part of the system. Inclusion of an LSTM allows the navigation agent to learn navigation policies and complex traversability cost functions and, furthermore, to retain memory of the knowledge learned in the past navigation episodes for reuse during new episodes. The latter capability allows expensive retraining to be avoided when the navigation environment is similar to those already explored by the agent and allows hidden states of the extremely large state space represented by a nonstructured or rough terrain to be recognized. We present the LSTM in the next section.

3. Long-short-term memory neural network

LSTM is a recurrent neural network architecture originally designed for supervised time-series learning. It addresses the problem that errors propagated back in time tend to vanish in multilayer neural networks (MLPs). Enforcing a constant error flow in constant error carousels (CEC) is a solution for vanishing errors [7].

These CECs are processing units having linear activation functions that do not decay over time. CECs can become filled with useless information if access to them is not regulated; therefore, specialized multiplicative units called input gates regulate access to the CECs. Further, their access to activation of other network units is regulated by multiplicative units called output gates. In addition, forget gates are added to CECs in order to reset information

that is no longer useful. A combination of a CEC and its input, output and forget gates is called a memory cell (**Figure 6**).

The activation updates at each time step t in this type of neural network are computed as follows. For the hidden unit activation y^h, the output unit activation y^k, the input gate activation y^{in}, the output gate activation y^{out} and the forget gate activation y^{φ}, we have

$$y^i(t) = f_i\left(\sum_m w_{im} y^m(t-1)\right) \tag{10}$$

where w_{im} is the weight of the connection from unit m to unit i. For the activation function f_i, the standard logistic sigmoid function for all units is chosen, except for output units, for which it is the identity function [7]. The CEC activation, also known as memory cell state, is calculated using

$$s_{c_j^v}(t) = y^{\varphi_j}(t) s_{c_j^v}(t-1) + y^{in_j}(t) g\left(\sum_m w_{c_j^v m} y^m(t-1)\right) \tag{11}$$

where g is a logistic sigmoid function scaled to the $[-2, 2]$ range and $s_{c_j^v}(0)$. Finally, the activation update for the memory cell output is calculated from

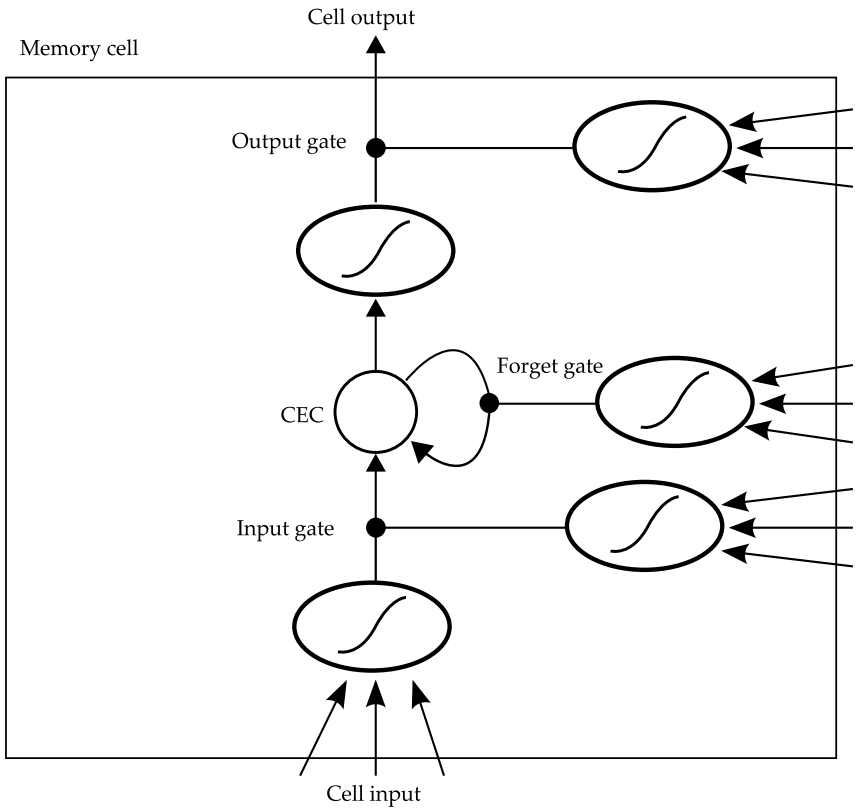

Figure 6. Graphic representation of a memory cell.

$$y^{c_j^v}(t) = y^{out}(t)h\left(s_{c_j^v}(t)\right) \tag{12}$$

The learning process implemented for LSTM in this paper is a variation of real-time recurrent learning (RTRL), as described in Ref. [7] which is a variation of [9]. In this variant, when the error arrives at a cell, it stops propagation further back in time. However, the error is used to update incoming weights when it leaves the memory cell through the input gate.

4. Reinforcement learning y long-short-term memory neural network

In order to teach the LSTM to navigate an unstructured terrain, RL was implemented as described in Ref. [7]. In this approach, an LSTM approximates the value function V of the RL algorithm, which teaches a robotic agent how to navigate a T-shaped maze environment.

This problem is a partially observable Markov decision process, in which the agent is unaware of the full state of the environment and must infer this information using current observations. In this study, these observations are the same feature vectors of the environment that were used for previous LEARCH experiments, and these vectors are the input for the LSTM.

The LSTM outputs represent the advantage values $A(s, a)$ of each action, where a is the action taken in state s. They are used to compute the value of the state $V(s) = \max_a A(s, a)$, which represents the action with the higher advantage value.

To perform weight updates, truncated backpropagation through time was implemented with RL. A function approximator's prediction error at time step t, $E^{TD}(t)$, is computed using Eq. (13) and is propagated one step back in time through all the units of the network, except for the CECs, for which the error is backpropagated for an indefinite amount of time [7]. Thus,

$$E^{TD}(t) = V\left(s(t) + \frac{r(t) + \gamma V(s(t+1)) - V(s(t))}{k} - A(s(t), a(t))\right) \tag{13}$$

where r is the immediate reward, γ is a discount factor in the $[0, 1]$ range and k scales the difference between the values of the optimal and suboptimal actions. It is worth mentioning that only the output associated with the executed action receives the error signal.

During the learning process, the agent can explore the environment using the state values; however, directed exploration (i.e. exploration for which a predictor is used to direct the exploration stage, so as to avoid clueless exploration of the entire state space) is important in order to learn complex terrain navigation. When an undirected exploration is conducted, RL tries every action in the same way over all states; however, in unstructured terrain, some states provide ambiguous information about the environment rendering it difficult for the agent to determine the state of the environment. Other states provide clear information; therefore, the agent must direct its exploration to discover the ambiguous states. In order to explore the environment, an MLP was implemented for directed exploration. This MLP input was the same as the LSTM, and the MLP objective was to predict the absolute value of the current temporal difference error, i.e. $E^{TD}(t)$. This aided prediction of which observations were

associated with a larger error. The desired MLP output was obtained using Eq. (14), and backpropagation was employed to train the MLP:

$$y_d^v(t) = \left|E^{TD}(t)\right| + \beta y^v(t+1) \tag{14}$$

The MLP output $y^v(t)$ is used as the temperature of the Boltzmann action selection rule, which has the form

$$\frac{e^{A(s,a)/y^v(t)}}{\sum_{b=1}^{n} e^{A(s,a)/y^v(t)}} \tag{15}$$

where n is the number of actions available to the agent.

The complete learning process and the manner in which the LEARCH and RL-LSTM systems are connected is shown in **Figure 7**. The entire process occurs offline. First, the LEARCH

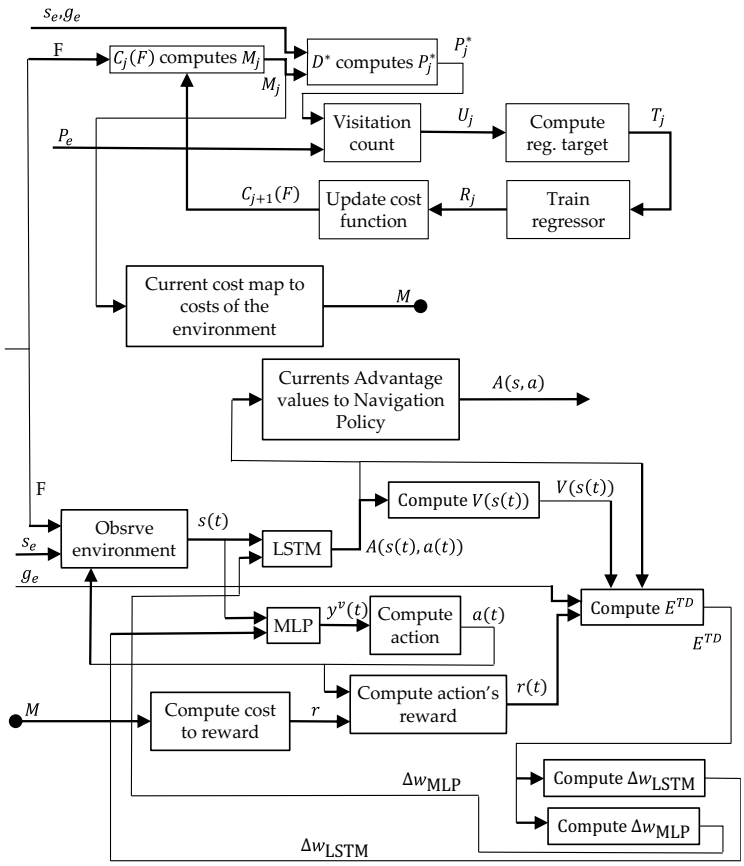

Figure 7. LEARCH-RL-LSTM system showing the manner in which the two systems are connected to train the LSTM. The entire process occurs offline. First, the LEARCH algorithm iterates until the required cost map M is obtained. Then, the RL-LSTM algorithm begins the process of training the LSTM using the costs converted into rewards r. The feature map F is obtained from the robotic agent and used by both systems.

Figure 8. (a) Example of a real environment modelled as a grid map. (b) Patches of terrain used for training are marked with an orange box.

algorithm iterates until the required cost map M is obtained. Then, the RL-LSTM algorithm begins the process of training the LSTM using the costs converted into rewards r. The feature map F is obtained from the robotic agent and used by both systems.

In order to prove the generalization and long-term memory capabilities of LSTM, training was performed using patches of terrain containing representative features of rough terrain. That is, an entire map is not used to train the LSTM (**Figure 8**). In this way, an efficient training phase is achieved by taking advantage of the above-mentioned capabilities. In the next section, we show the results of the experiments conducted to confirm these capabilities. In addition, we prove the efficacy of the LSTM for mapping tasks that require inference of hidden states, i.e. smoothing or noise recognition.

The LEARCH algorithm builds a cost function; however, as noted in Section 2, the cost function capability for generalization is limited and decays as the number of training paths grows. As the motivation for employing a cost function is to obtain the cost of traversing a patch of terrain so that the path planning system can compute the optimal path with the minimal traversal cost, we propose the extraction of terrain patches having descriptive characteristics of rough terrain for navigation. Hence, the traversal costs for these environment features can be determined using LEARCH, and the costs can be transformed to rewards for a RL algorithm [10].

5. Results

In this section, the results of the experimental tests are presented. Five environments were designed for navigation policy learning using the LEARCH-RL-LSTM system shown in **Figure 7**. Here, each environment was modeled as a grid, and each model was referred to as a map. Each map was a grid having dimensions of 20×20 cells. Further, each cell represented a patch of terrain, and this patch was represented by a vector of dimension 4, where each dimension was a scalar value representing the vegetation density, terrain slope, rock size or the presence of water.

Figure 9 shows in the lower right corner the color code used to illustrate the manner in which this environment was designed. Each environment differed by 5% from the previous one, i.e. 20% of the states in map 5 differed from those of map 1. These maps are shown in **Figure 9**.

Table 1 lists the results of experiments conducted using the LEARCH system alone to learn the navigation policies and cost functions of the five maps. In order to test the capability of LEARCH to reuse knowledge learned in previous navigation episodes, the following process was employed. Once LEARCH learned the navigation policies and cost function of map 1, this knowledge was used as initial knowledge to start navigation episodes involving the remaining maps. As is apparent from the first row of **Table 2**, it was not necessary to retrain the LEARCH row shows, and it was not necessary to retrain the LEARCH system to learn the demonstrated behavior and cost function of map 2. In other words, the LEARCH system could apply the knowledge learned from map 1 to map 2. However, this behavior did not occur for the other maps. For maps 3, 4 and 5, and when attempting to reuse the knowledge learned from map 1, it was necessary to retrain the LEARCH system. In these learning episodes, an increased number of iterations were necessary in order to acquire the new knowledge (as is apparent when **Table 1** is compared with row one on **Table 2**, it can be concluded that the previous knowledge learned using map 1 is even detrimental to the system performance when new

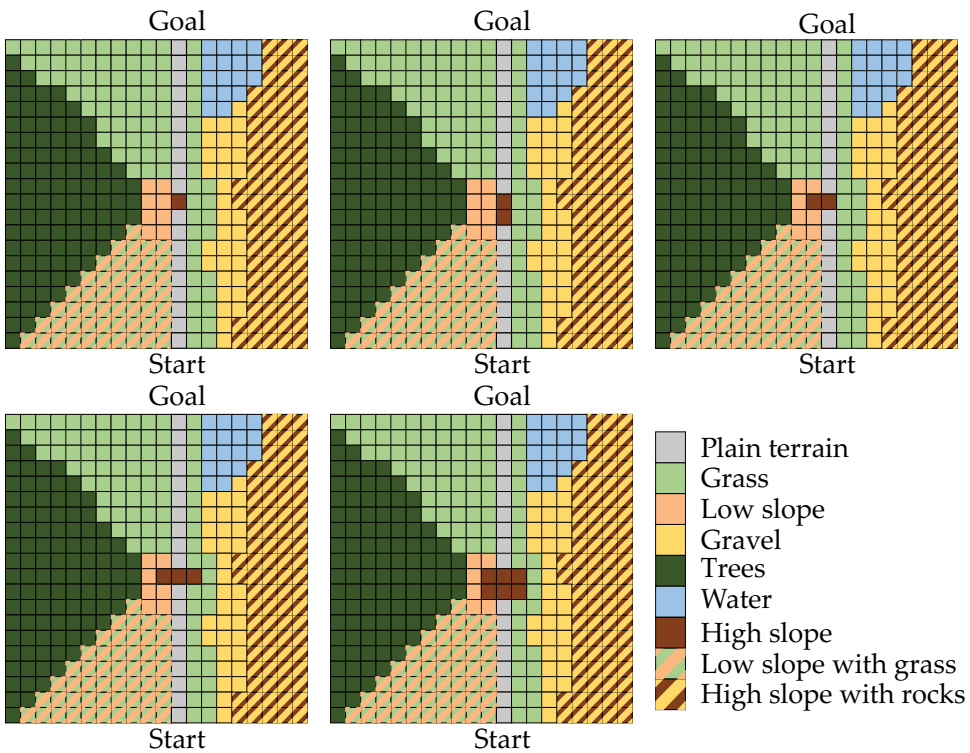

Figure 9. Maps used in experiments. Lower right corner of the second row: colour code used to represent environment features.

Environment	Map 1	Map 2	Map 3	Map 4	Map 5
Iterations	7	3	3	3	4

Table 1. Iterations needed to learn demonstrated behavior using LEARCH system.

Environment	Map 2	Map 3	Map 4	Map 5
Iterations LEARCH	0	5	4	7
Iterations LEARCH-RL-LSTM	0	0	0	0

Table 2. Iterations needed to learn demonstrated behavior using knowledge of map 1, for LEARCH and LEARCH-RL-LSTM system.

maps are processed, even if the new maps are very similar to map 1. Therefore, the LEARCH system was shown to have a very poor generalization capability.

When RL-LSTM was integrated with LEARCH to improve the capability for reusing knowledge learned from previous navigation episodes, there was no need to retrain the system. This is apparent from the second row of **Table 2**, where all the demonstrated behavior for maps 2 to 5 could be learned using the knowledge learned from map 1 only. It is important to note that, although the results were obtained from relatively small maps, it was necessary to retrain the cost function using LEARCH in each of these cases. Further, when LEARCH-RL-LSTM was employed, retraining was unnecessary when the patches of terrain were similar, because this system can generalize knowledge from previous navigation episodes. Note that, when the agent navigates in real time, even small retraining episodes are computationally expensive. Further, the agent is required to stop navigating until the retraining episode ends. However, for LEARCH-RL-LSTM, retraining is unnecessary when the environment is similar to those already known from previous navigation episodes.

Another set of environment maps was also used to test both algorithms. For these new environments, features that were not observed in previous scenarios were included. In this experiment, map 6 was the base of knowledge, and two new features were included in maps 7, 8 and 9. The states differed in the same way as in the previous experiment, with 5% of the states in each map being different from those of the previous maps. However, these differences included new features in order to simulate a dynamic environment, i.e. we simulate that the terrain of the map 6 suddenly changed when the agent navigates again on this map introducing new features on some cells of the grid of map 6. The maps used for this experiment are shown in **Figure 10**.

The LSTM used in these experiments was trained offline. During agent navigation, an efficient training episode was only executed if necessary, i.e. only if the action that LSTM learned to take is dangerous for the agent. These training episodes were efficient, because only a fraction of the environment was used (such as the patch of terrain shown in the lower right corner of **Figure 10** each time the robot encountered a new state or required navigation assistance.

http://dx.doi.org/10.5772/intechopen.71486

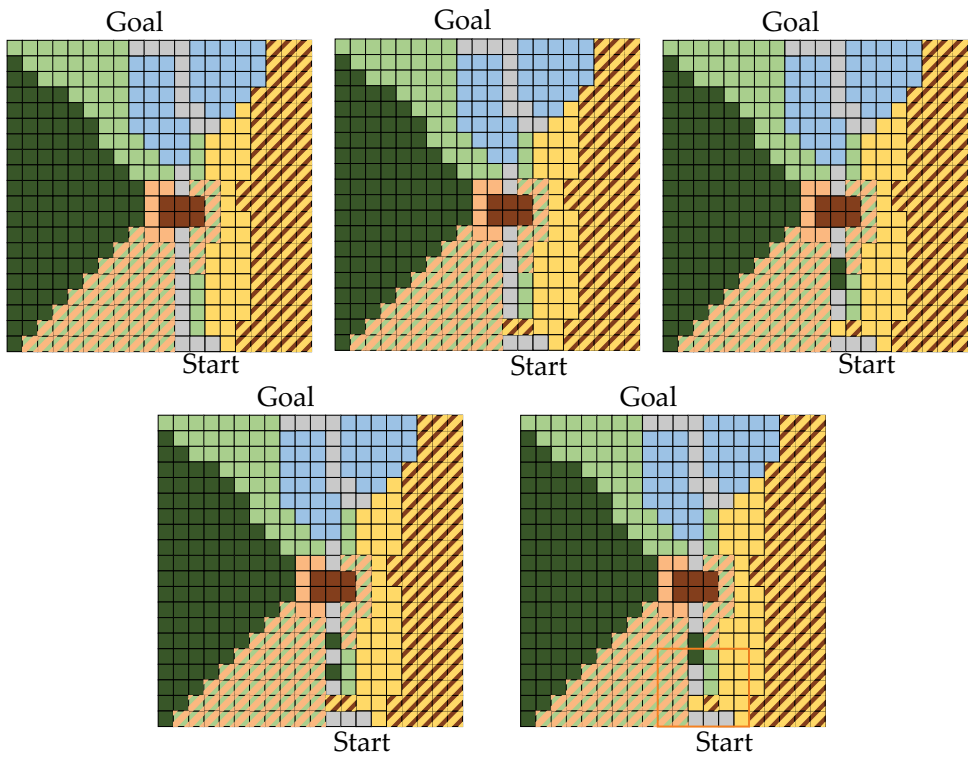

Figure 10. Maps used in second set of experiments to simulate dynamic environments. Lower right corner of the second row: sample of a map with the patch of terrain used for retraining marked by an orange box.

5.1. Noise tests

In the previous experiments, we assumed that the agent could infer the current state of the environment model based on the features observed by the agent. However, in a real scenario, the agent must infer the actual state via a perceptual system based on data obtained through noisy sensors such as cameras, a Global Positioning System (GPS) or LiDAR. In outdoor environments, two states (patches of terrain) can be very similar; however, the same action in these similar states could lead to different resultant actions. In case of noisy signals, one state could be interpreted as another similar state or, alternatively, as a new state that is not explicitly represented in the environment model, i.e. a hidden state.

To test these two systems in more realistic environment, a noise signal was induced to the inputs of both systems. A real uniform distribution bounded to a maximum of $[-1, 1]$ (20% of noise) was used. Then, several runs of each system were conducted with an initial limit of $[-0.1, 0.1]$ (2% of noise) and increments of $[-0.1, 0.1]$ in the noise signal, until the maximum limits where both systems failed to infer the real state for the agent were determined. **Tables 3** and **4** show the test results for both systems with noise; the noise range values are

Environment	Map 1	Map 2	Map 3	Map 4	Map 5
Noise-supported LEARCH	2%	2%	2%	2%	6%
Noise-supported LEARCH-RL-LSTM	10%	8%	10%	8%	12%

Table 3. Maximum noise supported by both systems in tests where the desired behavior could be reproduced with maps 1–5.

Environment	Map 6	Map 7	Map 8	Map 9
Noise-supported LEARCH	0%	0%	0%	0%
Noise-supported LEARCH-RL-LSTM	10%	8%	8%	8%

Table 4. Maximum noise supported by both systems in tests where the desired behavior could be reproduced with maps 6–9.

the maximum limits of the noise supported by the system using that map. Note that the results of the maps 6–9 yielded by the LEARCH system are omitted, because this system could not reproduce the desired behavior on these maps under the supplied noise levels.

6. Conclusion

LEARCH is an efficient method for learning a cost function that maps environment features to traversal costs and can then be used to navigate an unstructured terrain. However, as demonstrated by the experiments conducted in this work, this algorithm is incapable of reusing knowledge in an efficient manner. Indeed, zero knowledge is sometimes preferable to reusing previously learned knowledge.

We concluded that LEARCH cannot reuse knowledge because of a lack of memory; because of this lack of memory, the cost function cannot correlate knowledge learned in earlier training episodes with the new information provided by new environments; therefore, an LSTM was proposed. The LSTM can relate knowledge using memory cells, and this knowledge can be used to manage dynamic environments. This performance was demonstrated in experiment, where a dynamic environment was simulated through addition of new features that were not included in previous training episodes.

In addition, we implemented these two approaches to manage real scenarios in which noisy signals were present. The experiments showed that LEARCH-RL-LSTM can reproduce the desired behavior and navigate through the environment.

Author details

Nancy Arana-Daniel*, Roberto Valencia-Murillo, Alma Y. Alanís, Carlos Villaseñor and Carlos López-Franco

*Address all correspondence to: nancyaranad@gmail.com

Department of Computer Science, Universidad de Guadalajara, Guadalajara, Jalisco, México

References

[1] Silver D, Bagnell JA, Stentz A. Learning from demonstration for autonomous navigation in complex unstructured terrain. The International Journal of Robotics Research. 2010;**29** (12):1565-1592. DOI: 10.1177/0278364910369715

[2] Surger B, Steder B, Burgard W. Traversability analysis for mobile robots in outdoor environments: A semi-supervised learning approach based on 3D-lidar data. In: 2015 IEEE International Conference on Robotics and Automation (ICRA); 26–30 May 2015; Seattle, WA, USA. IEEE; 2015. p. 3941-3946. DOI: 10.1109/ICRA.2015.7139749

[3] Häselich M, Jöbgen B, Neuhaus F, Lang D, Paulus D. Markov random field terrain classification of large-scale 3D maps. In: 2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014); 5–10 Dec 2014; Bali, Indonesia. IEEE; 2014. p. 1970-1975. DOI: 10.1109/ROBIO.2014.7090625

[4] Kondo M, Sunaga K, Kobayashi Y, Kaneko T, Hiramatsu Y, Fuji H, Kamiya T. Path selection based on local terrain feature for unmanned ground vehicle in unknown rough terrain environment. In: 2013 IEEE International Conference on Robotics and Biomimetics (ROBIO); 12–14 Dec 2013; Shenzhen, China. IEEE; 2013. p. 1977-1982. DOI: 10.1109/ROBIO.2013.6739759

[5] Murphy L, Newman P. Risky planning on probabilistic Costmaps for path planning in outdoor environments. IEEE Transactions on Robotics. 2013;**29**(2):445-457. DOI: 10.1109/TRO.2012.2227216

[6] Valencia-Murillo R, Arana-Daniel N, López-Franco C, Alanís A. Rough terrain perception through geometric entities for robot navigation. In: 2nd International Conference on Advances in Computer Science and Engineering (CSE 2013); 1–2 Jul 2013; Los Angeles, CA, USA. Atlantis Press; 2013. DOI: 10.2991/cse.2013.69

[7] Bakker B. Reinforcement learning with long short-term memory. In: Advances in Neural Information Processing Systems 14. Cambridge: MIT Press; 2002. p. 1475-1482

[8] Kalman R. When is a linear control system optimal. Journal of Basic Engineering. 1964;**86**(1): 51-60

[9] Cortes C, Vapnik V. Support-vector networks. Machine Learning. 1995;**20**(3):273-297. DOI: 10.1007/BF00994018

[10] Sutton R, Barto A. Reinforcement Learning: An Introduction. Cambridge: MIT Press; 1998

Chapter 7

Path Planning Based on Parametric Curves

Lucía Hilario Pérez, Marta Covadonga Mora Aguilar,
Nicolás Montés Sánchez and
Antonio Falcó Montesinos

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.72574

Abstract

Parametric curves are extensively used in engineering. The most commonly used parametric curves are, Bézier, B-splines, (NURBSs), and rational Bézier. Each and every one of them has special features, being the main difference between them the complexity of their mathematical definition. While Bézier curves are the simplest ones, B-splines or NURBSs are more complex. In mobile robotics, two main problems have been addressed with parametric curves. The first one is the definition of an initial trajectory for a mobile robot from a start location to a goal. The path has to be a continuous curve, smooth and easy to manipulate, and the properties of the parametric curves meet these requirements. The second one is the modification of the initial trajectory in real time attending to the dynamic properties of the environment. Parametric curves are capable of enhancing the trajectories produced by path planning algorithms adapting them to the kinematic properties of the robot. In order to avoid obstacles, the shape modification of parametric curves is required. In this chapter, an algorithm is proposed for computing an initial Bézier trajectory of a mobile robot and subsequently modifies it in real time in order to avoid obstacles in a dynamic environment.

Keywords: path planning, mobile robots, parametric curves, Bézier curves

1. Introduction

In the last years, intelligent vehicles have increased their capacity up to the point of being able to navigate autonomously in structured environments. Implementations, such as Google [1] (with more than 700,000 hours of autonomous navigation in different scenarios), are an example of the effort made in this area. However, there is still a long way to go until we found real

autonomous cars on the roads, as there are both technical and legal problems involved [2, 3]. The intelligent system is composed of three different groups and subgroups: acquisition and perception, decision and actuation-control.

Although the vast majority of the literature often depicted the problems by focusing mainly on these groups or subgroups of processes, functionality in intelligent vehicles, or in mobile robots in general, cannot be conceived as composed of separate blocks, and therefore, a sufficiently efficient system can only be achieved if all the systems work in unison.

This chapter is devoted to the use of parametric curves in the field of robotics. Parametric curves are mainly used in the decision block when the path is defined. However, they are also employed in other blocks, and some of their properties are beneficial for other processes.

Focusing on the decision-making block, the "path planning" or the design of the path to follow has been the subject of study in the last decades, where many authors divide the problem into global and local path planning. On the one hand, the global path planning generates an overall path composed of a set of points to be followed, covering large distances and considering static obstacles in the environment. On the other hand, the local path planning constructs a short path with much more precision, even in continuous form, taking into account unexpected obstacles that may appear.

In general, path planning techniques can be grouped into four large groups: graph search, sampling, interpolating and numerical optimization, see [3]:

- *Graph search-based planners* search a grid for the optimal way to go from a start point to a goal point. Algorithms, such as Dijkstra, A-Start (A *) and its variants Dynamic A* (D*), field D*, Theta*, etc., have been extensively studied in the literature.
- *Sampling-based planners* try to solve the search problem restricting the computational time. The idea is to randomly explore/sample the configuration space, looking for connections between source and destination. The main problem is that the solution is suboptimal. The most common techniques are the probabilistic roadmap method (PRM) and the rapidly exploring random tree (RRT).
- *Interpolating curve planners* try to insert a new group of data within the previously defined data group. In other words, both graph search and sampling-based planners are global planners that provide a rough approximation of the solution. In this case, it is a matter of interpolating this group of points. At this stage, the design of the trajectory is when the properties of continuity, smoothness and geometrical restrictions of the vehicle, among others, intervene. Computer-aided geometric design (CAGD) techniques are generally used to smooth the gross path provided by the global planner. The use of lines and circles is usually employed as a first solution, with Dubin's curves defined when the vehicle moves forward and Reed and Sheep's curves when the vehicle moves backward. The clothoid appears as a solution to the discontinuity in curvature between the line and the circle since, by definition, it has a constant relationship between the length of the arc and its curvature. The polynomial curves are another alternative to the previous ones. The modification of its coefficients allows taking into account, among others, the adjustment of positions, curvature restrictions, etc.

- *Numerical optimization* is generally used to minimize or maximize a numerical function that depends on different variables such as smoothness, continuity, velocity, acceleration, jerk, curvature, etc.

In [3], the use of parametric curves is included in the category of *interpolating curve planners*. The most commonly used parametric curves in robotics are Béziers, B-splines, rational Bézier curves (RBCs), and non-uniform rational B-splines (NURBSs). A summary of their properties can be low computational cost, intrinsic softness, easy malleability through control points, and universal approximation. For these reasons, parametric curves are not only relevant as interpolators, but also recently they are being used in combination with many other algorithms that have effects on all the other blocks of an intelligent system [3].

The chapter is organized as follows. Section 2 provides a mathematical definition of the most used parametric curves as well as a description of their properties (Bézier, B-spline, RBC, and NURBS). Section 3 offers a state of the art of the use of parametric curves in robotics and an overview of current trends. Along the lines of the new trends in the use of these curves, Section 5 proposes a method of deformation of parametric curves aimed at modifying the trajectory in real time in order to avoid collisions. Section 6 presents the reader the conclusions.

2. Definitions: parametric curves

Curves in both space and plane are a part of the geometry necessary to represent certain shapes in different areas. Curves arise in many applications, such as art, industrial design, mathematics, architecture, engineering, etc.

2.1. Different ways of defining a curve. Advantages and disadvantages

There are different ways of defining a curve: implicit, explicit, and parametric.

2.1.1. Implicit and explicit expression of a curve

The coordinates (x, y) of the points of an implicitly defined plane curve verify that:

$$F(x,y)=0 \tag{1}$$

for some function F. If the curve is in R^3, then the curve must satisfy these two conditions simultaneously:

$$F(x,y,z)=0 \text{ and } G(x,y,z)=0 \tag{2}$$

The explicit representation of a curve clears one of the variables as a function of the other. In the plane, the coordinates (x,y) of the points in the curve explicitly defined satisfy either.

$$y = f(x) \text{ or } x = g(y) \tag{3}$$

In the case of a curve in the space, its explicit form could be defined as:

$$x = f(z) \text{ and } y = g(z) \tag{4}$$

2.1.2. Parametric expression of a curve

In this case, the coordinates of a parametric curve are expressed as a function of a parameter, for example, u. The definition of a curve defined in R^n could be done as in (5), where functions α_i are the coordinate functions or component functions. The image of $\alpha(u)$ is called the trace of α and α(u) is the parametrization of α.

$$\alpha : [a,b] \to \Re^n \; / \; \alpha(u) = (\alpha_1(u), \ldots, \alpha_n(u)); u \in [a,b] \tag{5}$$

Parametric curves are the most used in computer graphics and geometric modeling because the curve points are calculated in a simple way. In contrast, the calculation of the points through the implicit expression is much more complex.

Within the parametric curves, it is possible to differentiate between polynomial curves and rational curves. Polynomial curves are those whose component functions are polynomials, and rational curves are those expressed as the quotient of polynomials. The representation in the form of parametric curves allows a great variety of curves, some known, some strange, some complex and others surprising for their symmetry and beauty.

The advantageous properties of the parametric curves that make them widely used are intuitivity, flexibility, affine-invariant, fast computation, and numerical stability.

In order to model complicated shapes or surfaces, it is necessary to introduce a way of representing curves based on a polygon. From this idea, the most used parametric curves arise in computer-aided geometric design (CAGD): Bézier, B-splines, RBC, and NURBS. **Figure 1** shows a schematic of the most important curves in CAGD. It can be seen how NURBS are the most general curves, and Bézier are the most particular ones. Among them, Bézier is the simplest, possessing properties that make them be the most extensively used.

2.2. Most common parametric curves: Bézier, B-spline, NURBS, and RBC

2.2.1. Bézier curves

Bézier curves arose as a result of the car modeling in both Renault and Citroën companies, by the engineers Pierre Bézier and Casteljau. The simplicity in the manipulation of these curves makes their use and application widespread.

The popularity of the Bézier curves is due to their numerous mathematical properties that facilitate their manipulation and analysis. Moreover, their use does not require great mathematical knowledge, which is very interesting for designers who shape objects.

http://dx.doi.org/10.5772/intechopen.72574

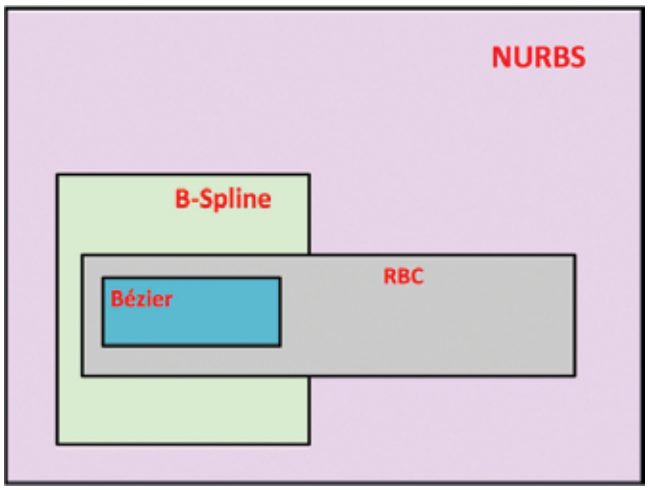

Figure 1. Classification of the most important curves in CAGD.

A Bézier curve of degree n is specified by a sequence of $(n + 1)$ control points, and its explicit expression is (6). The polygon that joins the control points is called the control polygon, and the functions or bases used are the Bernstein polynomials $B_{i,n}(u)$, defined in (7).

$$\alpha(u) = \sum_{i=0}^{n} P_i B_{i,n}(u); u \in [0,1] \tag{6}$$

$$B_{i,n}(u) = \binom{n}{i} u^i (1-u)^{n-i}; i = 0, \ldots, n \tag{7}$$

The dimension of the vector containing the control points is related to the dimension of the space where the curve is represented.

2.2.2. Rational Bézier curves

A conic is a curve obtained as the intersection of a plane with the surface of a double cone. There are three types of irreducible conics: hyperbola, parabola, and ellipse. Parabolas can be parameterized by polynomial functions, but hyperbolas and ellipses need rational functions such as RBC. The explicit definition of an RBC is (8), where P_i are the control points, $B_{i,n}(u)$ are the Bernstein bases, and ω_i are weights associated with each control point. These weights allow a new way of modifying the curve.

$$\alpha(u) = \frac{\sum_{i=0}^{n} \omega_i P_i B_{i,n}(u)}{\sum_{i=0}^{n} \omega_i B_{i,n}(u)}; u \in [0,1] \tag{8}$$

2.2.3. B-spline curves

B-splines are polynomial curves defined in pieces, continuously differentiable up to a prescribed order. The name spline is a word that means "elastic slats". These slats were used by craftsmen to create curves describing the surfaces to be built, such as boat hulls and aircraft fuselages. Constrained by weights, these elastic slats or splines assume a shape that minimizes their elastic energy.

B-spline curves were developed to overcome the limitations of Bézier curves: the need for a local control of the curve, the difficulty in imposing C2 continuity and the fact that a number of control points of a Bézier curve imposes its degree.

Analogous to the definition of a Bézier curve, a B-spline curve of degree k (or $k + 1$ order) is expressed in (9) as an affine combination of certain control points P_i, where $N_{i,k}$ are polynomial functions by pieces with finite support of order k (degree k-1, meaning that they are zero out of a finite interval) that satisfy certain conditions of continuity. Each of these functions can be calculated using the Cox-de-Boor recursive formulas.

$$\alpha(u) = \sum_{i=0}^{n} \mathrm{P}_i N_{i,k}(u) \tag{9}$$

B-splines can be defined by a recurrence relationship; simplicity is considered a double infinite sequence of simple nodes such that for all i. B-splines are then defined through the following recurrence relationship.

For the sake of simplicity, a double infinite sequence of simple nodes a_i is considered such that $a_i < a_{i+1}$ for all i. Then, the B-splines $N_{i,k}$ are then defined through the recurrence relationships (10) and (11).

$$N_{i,0}(u) = \begin{cases} 1 \text{ if } u \in [a_i, a_{i+1}[\\ 0 \text{ otherwise} \end{cases} \tag{10}$$

$$N_{i,k}(u) = \frac{t - a_i}{a_{i+k-1} - a_i} N_{i,k-1}(u) + \frac{a_{i+k-t}}{a_{i+k} - a_{i+1}} N_{i+1,k-1}(u) \tag{11}$$

2.2.4. Non-uniform rational B-spline curves

Rational B-spline curves are obtained in a similar way as the RBCs from Bézier curves. The definition of a NURBS curve is:

$$\alpha(u) = \frac{\sum_{i=0}^{n} \mathrm{P}_i \omega_i N_{i,k}(u)}{\sum_{i=0}^{n} \omega_i N_{i,k}(u)} \tag{12}$$

PARAMETRIC CURVES IN CAGD	Polinomials curves (Non rationals)		Rationals curves (conics)	
	Bézier	B-Spline	Rational Bézier	NURBS
Convex Hull Property	Yes	Yes (only local)	Yes (if the weights are positive)	Yes (only local if the weights are positive)
Interpolation of first and last control point	Yes	No	Yes	No
Tangency to the control polygon first and last point	Yes	No	Yes	No
Invariant under affine transformations	Yes	Yes	Yes	Yes
Recurrent bases	No	Yes	No	Yes
Local control	No	Yes	No	Yes
Global control	Yes	No	Yes	No

Table 1. Comparison of the properties of parametric curves.

2.3. Comparison of properties

When dealing with curves, their representation is important, but their shape manipulation is a key factor in their usability. The object to be modeled will determine the type of parametric curve chosen, depending on the properties required. In **Table 1**, we can see a comparison of the properties of the parametric curves in CAGD. In the following section, some of the most relevant works in mobile robots using parametric curves are described.

3. Use of parametric curves in robotics: state of the art

3.1. Generation of trajectories of mobile robots through parametric curves

Predicting the movement of a robot is important as it implies the computation of a proper path that meets the kinematic and dynamic properties of the robot. Simply moving a mobile robot from an initial position (x_i, y_i, θ_i) to a final position (x_g, y_g, θ_g) safely implicates many research fields, which are involved in the generation of efficient path planning algorithms.

Many researchers consider parametric curves very useful in the construction of trajectories of wheeled robots, due to their advantageous properties able to improve trajectories produced by path planning techniques.

3.2. Trajectories of mobile robots defined by Bézier curves

The first relevant publications in robotics using Bézier curves are published in 1997 and 1998 [29, 30]. These works combine path planning and reactive control for a non-holonomic mobile robot introducing the concept of "Bubble Band" (bubble path). With an appropriate metric, bubbles are connected with Bézier curves, generating a path. These bubbles are the maximum free space reachable in any direction without risk of collision. This is due to the property of the convex hull and implies that if the control points are within the bubble, then the path approximation remains within the bubble. The planner, using a model of the environment, generates an initial path connecting the start and goal positions that may not be adequate. Next, the proposed algorithm generates a sequence of bubbles connecting both ends and replacing the original path, the *bubble band*. This band is exposed to the forces in the environment, and as a consequence, the band is modified.

In 2001, the concept of "bubble band" is used in [31]. In this case, dynamic obstacles are introduced in the environment. Simultaneously, in [32], also Bézier curves are used for local path planning. An initial path is computed using the generalized Voronoi graph (GVG) theory, which is mildly deformed maximizing the evaluation of a function. Candidates obtained as smooth paths are expressed with Bézier curves.

In 2003, a touchscreen was introduced in [33] to control a mobile robot, avoiding obstacles in real time. In this work, two algorithms are developed: the first one extracts a succession of important points, and the second one generates a path using cubic Bézier curves.

In 2007, the work in [34] introduces Bézier curves in cooperative collision avoidance for several mobile non-holonomic robots and is based on the previous contributions [35, 36]. Two tasks are developed: first, path planning based on Bézier curves for each individual robot in order to obtain its final position and, second, computation of an optimal path that minimizes a "penalty" function that accounts for the sum of the maximum times subject to the distances between the robots.

In 2008, [37] presents a preliminary framework that generates space trajectories for multiple unmanned aerial vehicles (UAVs) using 3D Bézier curves. The algorithm solves a constrained optimization problem in order to generate the trajectories. In this case, the optimization function penalizes an excessive length, as the shortest path is required, and the restrictions are the distances between the multiple UAVs. The system is non-linear, and numerical methods are applied to solve it.

It is worth mentioning the work of Choi et al. [38–46] related to the computation of trajectories of mobile robots designed from Bézier curves. In many of the publications, a constrained optimization problem is raised, where the function to be optimized is the curvature of a Bézier curve.

Finally, [47] presents a methodology based on the variation of the RRT that generates suitable trajectories for autonomous vehicles with holonomic constraints in environments with

obstacles. This algorithm is based on the use of seventh-order Bézier curves that connect the vertices of the tree. In this way, the generated paths meet the main kinematic constraint of the vehicle: the smoothness of the acceleration is guaranteed for the entire path by controlling the values of the curvature of the endpoints of each Bézier curve composing the tree. The proposed algorithm provides a rapid convergence to the final result. In addition, the number of vertices of the tree is reduced because the method allows the connections between the vertices of the tree with an unlimited range. The properties of seventh-order Bézier curves are also used to avoid static obstacles in the environment. This method was simulated with a small UAV. Since then, B-splines and Beziers curves have been used to generate search trees by a large number of researchers, see [7].

Recent efforts are being made to merge Bézier curves with numerical optimization, [4, 5]. In these works, a teleoperation is carried out where the operator indicates some points. The proposed algorithm calculates the path to continuity of curvature C1 and C2. In [6], something similar is proposed: nodes/points are initially generated between the start and the goal (collision-free) and then are joined by cubic Bézier curves with curvature constraints. Finally, cubic Bézier curves are used in [8] to solve the problem of roundabouts for automated vehicles: entry, departure, and crossing.

3.3. Trajectories of mobile robots defined by B-spline

In 1989, B-spline curves were incorporated in the design of robot trajectories. In [13], segments were added with the aim of generating the entire path near the desired one. This new trajectory did not go through the exact points. Later, in 1994, the work in [14] used B-splines for path planning but adding a temporal variable. In this case, the speed of the robot was controlled by the same B-spline. The same year, in [15], a fuzzy controller is designed to emulate spline curves for generating smooth motion trajectories. In 1999, the work [16] also used a B-spline curve to calculate the trajectory of a mobile robot by generating many points from a spline for the robot to follow them in the form of succession. Additionally, in [17], kinematic constraints were introduced in the path planning using B-spline curves to find the optimal temporal trajectory in a static environment.

Lately, in 2007, the works [18–20] developed a method to solve the path planning problem using cubic splines to avoid the obstacles. This method iteratively refined the path to be followed in order to obtain in real time a collision-free feasible path in unstructured environments. In [20], the path planning implementation based on B-spline is detailed. The use of splines allows to restrict the polynomials since the first derivative of $P_1,\ldots, P_{n\text{-}1}$ is continuous across the entire boundary. In addition, some constraints can be introduced on the first and last points to force a particular value of the derivative. These characteristics of the splines offer many advantageous properties to plan a suitable path. If a value of the derivative is imposed, a path can be generated starting from a specific position and having a direction imposed by the value of the derivative. Therefore, they can be generated and initialized from the current position and direction of the vehicle. The first derivative is proportional to the direction of the vehicle, then a non-continuous derivative could be obtained and, as a consequence, a non-feasible path for that type of vehicle. As the second derivative is proportional to the direction

of the angle, some discontinuities could force the vehicle to stop at each control point to adjust its direction.

B-splines curves allow an easy construction of smooth paths through control points. In order to avoid obstacles, control points are introduced near them, and methods are developed to move these control points away from the obstacles and move them to the free space.

Earlier methods also worked with splines to generate smooth paths also avoiding the surrounding obstacles [20, 21]. Nevertheless, these previous methods had a high computational cost when evaluating the overall path. In [18], the computational time and the viability of one of these algorithms are analyzed, since it is executed with an iterative method. Monte Carlo simulations indicate a high degree of success for complex environments. The running time is also measured and increases with the complexity of the environment. Finally, in [19], experimental results are provided. The main disadvantage of this algorithm is that the obstacle-free path is computed by means of an iterative method. Thus, the computational time will always increase with respect to other non-iterative methods.

A large number of researchers have also used parametric curves, and particularly B-splines and Béziers, to generate search trees as in [7].

3.4. Trajectories of mobile robots defined by NURBS

This type of parametric curve is used in the reconstruction of trajectories with the aim of generating smooth paths that approximate the real movement of the robot. In [22], the advantages and disadvantages of the NURBS curves are highlighted, providing a detailed study of their properties. In the field of robotics, the work [23] highlights advantageous properties of NURBS for path planning in both 2D and 3D.

In other works, such as [24–26], NURBS curves approximate or describe the path described by a robot arm PUMA 560. Programming by Demonstration is used to program the behavior of the robot, a good solution to automatically transfer the human knowledge to a robot. However, the NURBS trajectory does not guarantee the obstacle avoidance.

More recently, in [9–12] a predefined NURBS curve is used to improve its properties adjusting the weights.

3.5. Trajectories of mobile robots defined by RBC

In [27], an off-line methodology is presented to approximate a Clothoid (Fresnel integrals) to an RBC. Subsequently, [28] presents a method to obtain trajectories in real time with Clothoids. To do this, two steps are involved: the off-line definition of approximations of Clothoids with RBCs and the generation of online paths by scaling, rotating and moving the previous off-line curves. One of the advantages of this method is the off-line calculation since it considerably reduces the computational time. Throughout the process, the weight coefficients and control points remain invariant. In this work, it is guaranteed that an RBC has the same behavior as a Clothoid using a low order for the curve.

3.6. Current trends in the use of parametric curves in robotics

This comprehensive study of the use of the parametric curves evidences its importance in the design of trajectories of a mobile robot. They are not only used for interpolating points in the global map but also being integrated into global planners and in numerical optimizations. Although non-rational curves have a lower approximation capacity, researchers prefer them for their simplicity and easy manipulation. Among them, we must highlight the Bézier curves, which are the most used.

However, when the parametric curve is used as an approximation, the use of rational curves is significantly greater, as in the approximation to the clothoid and the circle. Recently, predefined rational curves are being used, where only the weights are modified. This can transform rational curves into manageable curves in comparison to non-rational curves.

Along the lines of merging the use of parametric curves with other types of algorithms in an intelligent navigation system, it is not only important to define the path of the robot, but also to avoid obstacles in the environment. Consequently, the initial trajectory must be modified in real time so that the mobile robot avoids the possible dynamic obstacles that may appear. In this sense, the Bézier trajectory deformation (BTD) algorithm, described in the next section, introduces the possibility of deforming a Bézier curve through a vector field, which can be used in mobile robotics. The temporal parameter is introduced in the Bézier curve to transform it into a path and a vector field is needed to modify the initial path.

4. Properties of parametric curves and its applications in robotics

In mobile robotics, two main needs have arisen when dealing with path planning of a mobile robot: definition of the initial path to follow and the possibility of modifying it in the presence of dynamic obstacles.

In the next paragraphs, the BTD algorithm is described [48, 49], which solves the abovementioned needs. It offers the possibility of defining the trajectory of a mobile robot through a Bézier curve and then modifies it by means of the repulsive forces derived from a predictive potential field (PF) method. Reactive methods or potential field methods generate obstacle-free paths for the robot. In these methods, the movement of the robot is determined by repulsive forces associated with obstacles and attractive forces associated to the goal position of the mobile robot. In this work, the potential field projection method (PFP) has been used [50, 51].

The set of discrete points provided by the posture prediction of the mobile robot is considered as *initial points* S_i of the original Bézier curve. These points belong to a reference path in the BTD algorithm. Subsequently, the set of repulsive forces obtained by the PFP is transformed into displacements by a dynamic particle model, which generates *endpoints* T_i that determines the modification of the original Bézier trajectory with the BTD. A modified Bezier trajectory free of obstacles is obtained that passes through the endpoints, as displayed in **Figure 2**.

The definition of the BTD algorithm requires two steps:

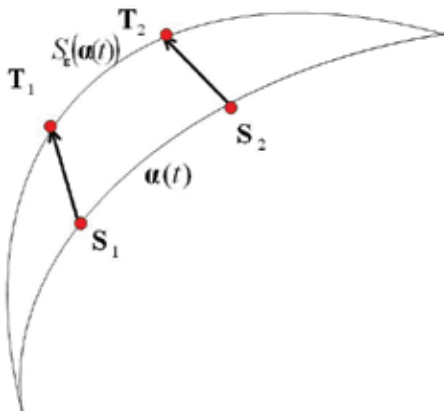

Figure 2. Deformation of a Bézier trajectory through a field of vectors.

a. *Definition of the trajectory using a Bézier curve*

A Bézier curve has a non-dimensional intrinsic parameter, u, as defined in (5). Since the Bézier curve represents the path of the robot, the intrinsic parameter must be defined as a temporal variable so that the position of each curve (robot position) is associated with an instant of time $t \in [t_0, t_f]$, where t_0 and t_f represent the initial and final times of the trajectory, respectively. The definition of the initial Bézier trajectory is (13), where n is the order, P_i are the control points and $B_{i,n}(t)$ are the Bernstein bases defined in (14). To avoid loops in the Bézier curve, second-order curves are used.

$$\alpha(t) = \sum_{i=0}^{n} \mathrm{P}_i \cdot B_{i,n}(t), t \in [t_0, t_f] \tag{13}$$

$$B_{i,n}(t) = \binom{n}{i} \left(\frac{t_f - t}{t_f - t_0}\right)^{n-i} \left(\frac{t - t_0}{t_f - t_0}\right)^{i}; i = 0, \cdots, n \tag{14}$$

The initial Bézier trajectory will be deformed in order to avoid the surrounding obstacles, by modifying the position of the control points from the initial position to the new one imposed by the PFP obstacle avoidance algorithm. The displacement of each control point $P_i$is denoted as ε_i, so that the vector $\varepsilon = [\varepsilon_0, \cdots, \varepsilon_n]$ is the displacement of all the control points defining the Bézier trajectory, also known as the perturbation vector of the deformed curve. The new modified Bézier trajectory $S_\varepsilon(\alpha(t))$ is defined in (15) and, consequently, the optimizing function used to solve the problem is defined as (16), where the vector ε is computed as in [49].

$$S_\varepsilon(\alpha(t)) = \sum_{i=0}^{n} (\mathrm{P}_i + \varepsilon_i) \cdot B_{i,n}(t), t \in [t_0, t_f] \tag{15}$$

$$\min_{\varepsilon} \int_{t_0}^{t_f} \left| S_\varepsilon(\alpha(t)) - \alpha(t) \right|_2^2 dt \tag{16}$$

This objective function minimizes changes in the shape of the initial Bézier trajectory as it minimizes the distance between the original Bézier trajectory and the modified one. This definition is suitable for holonomic mobile robots since the original path has been generated by a global path planner, and the original path is assumed to be already optimal.

b. *Number of Bézier curves*

High order Bézier curves are numerically unstable and, for that reason, in order to generate a complete Bézier trajectory the concatenation of k curves is required. Therefore, the optimization function (16) is replaced by (17), where $1 \leq l \leq k$, α_l is every Bézier trajectory (l), $S_\varepsilon(\alpha_l)$ is the modified Bézier trajectory, $\left[t_0^{(l)}, t_f^{(l)}\right]$ are the initial and end instants of the Bézier trajectory (l), and $\varepsilon^{(l)}$ is the perturbation vector of the modified curve (l).

$$f(\varepsilon) = \min_{\varepsilon^{(1)} \cdots \varepsilon^{(k)}} \sum_{l=1}^{k} \int_{t_0^{(l)}}^{t_f^{(l)}} \left\| S_\varepsilon\left(\alpha_l(t)\right) - \alpha_l(t) \right\|_2^2 dt \tag{17}$$

The number of repulsive forces depends on the order of the Bézier trajectory: $n_{(l)} - 1$.

c. The constraints of the optimization problem are:

i. *The mobile robot must follow a collision-free path:* The modified Bézier path must pass through the endpoints, so the robot does not collide with the obstacles the environment. The vectors joining the initial and end points are the repulsive forces obtained by the PFP method. The equation of this constraint is (18).

$$g_1 = \sum_{l=1}^{k} \sum_{j=1}^{n_l} \left\langle \lambda, T_j^{(l)} - S_\varepsilon\left(\alpha_l\left(t_j^{(l)}\right)\right) \right\rangle \tag{18}$$

ii. *The robot trajectory must be smooth:* this constraint implies imposing continuity and derivability in the joint points of two curves, expressed by Eq. (19).

$$\begin{aligned} g_2 &= \sum_{l=1}^{k-1} \left\langle \lambda, S_\varepsilon\left(\alpha_l\left(t_f^{(l)}\right)\right) - S_\varepsilon\left(\alpha_l\left(t_0^{(l+1)}\right)\right) \right\rangle \\ g_3 &= \sum_{l=1}^{k-1} \left\langle \lambda, S_\varepsilon\left(\alpha_l'\left(t_f^{(l)}\right)\right) - S_\varepsilon\left(\alpha_l'\left(t_0^{(l+1)}\right)\right) \right\rangle \end{aligned} \tag{19}$$

iii. Continuity between the present and future positions must be ensured: tangency must be maintained between the original Bézier trajectory and the deformed Bézier trajectory at the initial and end points of the trajectory. The equation is (20).

$$g_4 = \left\langle \lambda, \alpha_1'\left(t_0^{(1)}\right) - S_\varepsilon\left(\alpha_1'\left(t_0^{(1)}\right)\right) \right\rangle + \left\langle \lambda, \alpha_k'\left(t_f^{(k)}\right) - S_\varepsilon\left(\alpha_k'\left(t_f^{(k)}\right)\right) \right\rangle \tag{20}$$

With the objective function and the constraints, the Lagrangian function (21) is defined. In order to calculate the stationary points, the partial derivatives of the Lagrangian function are calculated and canceled, and a system of linear equations is obtained. The solution of this linear system is the perturbation vector of each control point in order to obtain the Bézier trajectory. In-depth information about the linear system obtained is described in [48].

$$L(\varepsilon, \lambda_i) = f(\varepsilon) - g_1 - g_2 - g_3 - g_4 \tag{21}$$

4.1. Numerical simulation

In our numerical example, it is used Bézier trajectories of second order to avoid loops in the trajectory. For that reason, the number of Bézier curves will be equal to the number of repulsive forces generated with the selected predictive PF technique. One vector is placed per Bézier curve. To develop, the BTD algorithm is necessary to follow these two steps:

1. Calculation of the control points from the prediction horizon generated with the PFP

The control points are uniformly distributed throughout the prediction horizon generated by the PFP method. The model has been developed for holonomic robots, and therefore, the prediction of future positions provides a straight line. In this case, the control points calculated through the formulation are obtained in **Table 2**.

2. Location of the repulsion forces on the Bézier curve

The control points of the Bézier curve are uniformly distributed, and the repulsion forces obtained with the PFP method are placed at the midpoint of each curve, except for the first and the last curves where they are placed at the first and last points, respectively.

Control Points of 1st curve	Control Points of 2nd curve	Control Points for the j-th curve
$P_0^{(1)} = \hat{x}(1)$	$P_0^{(2)} = P_2^{(1)}$	$3 \le j \le k-2$
$P_1^{(1)} = \hat{x}(1) + \frac{1}{3}\cdot\left(\hat{x}(2) - \hat{x}(1)\right)$	$P_2^{(2)} = \hat{x}(2) + \frac{1}{2}\cdot\left(\hat{x}(3) - \hat{x}(2)\right)$	$P_0^{(j)} = \hat{x}(j-1) + \frac{1}{2}\cdot\left(\hat{x}(j) - \hat{x}(j-1)\right)$
$P_2^{(1)} = \hat{x}(1) + \frac{2}{3}\cdot\left(\hat{x}(2) - \hat{x}(1)\right)$	$P_1^{(2)} = P_0^{(2)} + \frac{1}{2}\cdot\left(P_2^{(2)} - P_0^{(2)}\right)$	$P_1^{(j)} = P_0^{(j)}$
		$P_2^{(1)} = \hat{x}(j)$

Control Points next to the last curve	Control Points of the last curve
$P_0^{(k-1)} = \hat{x}(k-2) + \frac{1}{2}\cdot\left(\hat{x}(k-1) - \hat{x}(k-2)\right)$	$P_0^{(k)} = P_2^{(k-1)}$
$P_2^{(k-1)} = \hat{x}(k-1) + \frac{1}{3}\cdot\left(\hat{x}(k) - \hat{x}(k-1)\right)$	$P_1^{(k)} = \hat{x}(k-1) + \frac{2}{3}\cdot\left(\hat{x}(k) - \hat{x}(k-1)\right)$
$P_1^{(k-1)} = P_0^{(k-1)} + \frac{1}{2}\cdot\left(P_2^{(k-1)} - P_0^{(k-1)}\right)$	$P_2^{(k)} = \hat{x}(k)$

Table 2. Calculation of control points from the prediction horizon: $\hat{x}$ is the vector containing the future trajectory and $P_i^{(j)}$ is the *i*-th control point of the j-th curve.

http://dx.doi.org/10.5772/intechopen.72574

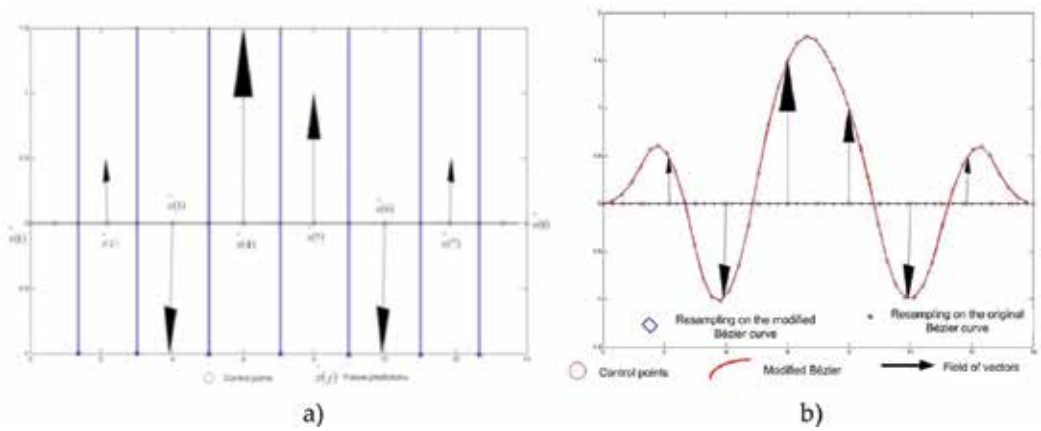

Figure 3. (a) Control points and future predictions of Bézier trajectory and (b) deformation of eight concatenated Bézier curves.

In **Figure 3(a)**, an example is shown, where a straight line represents the predicted optimal trajectory for a mobile robot obtained with the PFP algorithm. The control points needed to obtain the Bézier curves are displayed with red circles. The repulsive forces are placed in the proper positions of the predicted path. In this graphic example, there are eight points in the prediction horizon, and consequently, eight Bézier curves are concatenated in a straight line. The time devoted to perform trajectory is defined by the PFP prediction and has to be of 14 seconds. The time intervals corresponding to each curve, respectively, are [0,1.33], [1.33,3], [3,5], [5,7], [7,9], [9,11], [11,12.66], [12.66,14]. The representation of the resampling for the concatenation of eight Bézier curves is represented in **Figure 3(b)**.

5. Conclusion

This chapter details a comprehensive study of the use of parametric curves in the design of trajectories for holonomic and non-holonomic mobile robots. First, a brief introduction of the mathematical formulation and properties of the different curves is presented. Second, an exhaustive revision of literature regarding the use of parametric curves in path planning for mobile robots is developed. Third, a detailed description of the available techniques for path planning with parametric curves is presented, thoroughly describing the most important ones. Finally, an in-depth comparison is carried out between the different techniques of path deformation using Bézier curves, with their advantages and drawbacks. The Bézier curves are extensively used in these applications due to the simplicity of its definition and its easy handling and manipulation. The last section describes how to merge artificial potential field methods with Bézier curves as a solution for modifying a predefined trajectory in real time. Future works are related to the inclusion of other parametric curves, such as B-splines, RBC, and NURBS, in the proposed algorithm.

Author details

Lucía Hilario Pérez[1]*, Marta Covadonga Mora Aguilar[2], Nicolás Montés Sánchez[1] and Antonio Falcó Montesinos[1]

*Address all correspondence to: luciah@uchceu.es

1 Departamento Matemáticas, Físicas y Ciencias Tecnológicas, Universidad Cardenal Herrera-CEU, CEU Universities, Alfara del Patriarca, Spain

2 Department of Mechanical Engineering and Construction, Universitat Jaume I, Castellón, Spain

References

[1] https://www.google.com/selfdrivingcar/

[2] Van Schijndel-de Noóij M, et al. Definition of neccesary vehicle in infraestructure systems for automated driving. European Commission, Brusels, Belgium. SMART 2010/0064, 2011

[3] Gonzalez D, Perez J, Milanes V, Nashashibi F. A review of motion planning techniques for automated vehicles. IEEE Transactions on Intelligent Transportation Systems. 2016;**17**(4). DOI: 10.1109/TITS.2015.2498841

[4] Simba KR, Uchiyama N, Sano S. Real-time smooth trajectory generation for non-holonomic mobile robots using Bézier curves. Robotics and Computer-Integrated Manufacturing. 2016;**41**:31-42. doi.org/10.1016/j.rcim.2016.02.002

[5] Simba KR, Uchiyama N, Sano S. Real-time trajectory generation for mobile robots in a corridor-like space using Bézier curves, In: IEEE/SICE International Symposium on System Integration (SII). 2013. pp. 37-41. http://dx.doi.org/10.1109/SII.2013.6776639

[6] Cimurs R, Hwang J, II.H.Suh. Bézier curve-based smoothing for path planner with curvature constraints. IEEE International conference on Robotic computing. 2017. DOI: 10.1109/IRC.2017.13

[7] Elbanhawi M, Simic M, Jazar R. Randomized bidirectional B-spline parameterization motion planning. IEEE Transactions on Intelligent Transportation Systems. 2016;**17** (2):406-419. DOI: 10.1109/TITS.2015.2477355

[8] Gonzalez D, Perez J, Milanes V. Parametric-based path generation for automated vehicles at roundabouts. Expert Systems with Applications. 2017;**71**:332-341. DOI: 10.1016/j.eswa.2016.11.023

[9] Singh AK, Aggarwal A, Vashisht M, Siddavatam R. Robot motion planning in a dynamic environment using offset NURBS. IEEE ICIT. 2011:312-317. DOI: 10.1109/ICIT.2011.5754393

[10] Xidias EK, Aspragathos NA. Continuous curvature constrained shortest path for a car-like robot using S-Roadmaps. IEEE MED. 2013:13-18. DOI: 10.1109/MED.2013.6608692

[11] Jalel S, Marthon P, Hamouda A. NURBS based multi-objective path planning. In: Carrasco-Ochoa J, Martínez-Trinidad J, Sossa-Azuela J, Olvera López J, Famili F. eds. Pattern Recognition. MCPR 2015. Lecture notes in computer science, vol. 9116. Springer, Cham. 2015. doi.org/10.1007/978-3-319-19264-2_19

[12] Jalel S, Marthon P, Hamouda A. A new path generation based on accurate NURBS curves. International Journal of advanced Robotic systems. 2017;**13**(2), DOI:10.5772/63072

[13] Komoriya K, Tanie K. Trajectory design and control of a wheel-type mobile robot using B-spline curve. In Int. Workshop on Intelligence Robots&Systems. pp. 389-405. 1989. DOI: 10.1109/IROS.1989.637937

[14] Vázquez GB, Sossa AH, and Diaz de Leon S. Auto guided vehicle control using expanded time B-spline. In: IEEE International Conference on Systems, Man and Cybernetics. 1994;**3**:2786-2791. DOI: 10.1109/ICSMC.1994.400295

[15] Zhang J, Raczkowsky J, Herp A. Emulation of spline curve and its applications in robot motion control. In IEEE Int. Conference on Fuzzy Systems. 1994;**2**:831-836. DOI: 10.1109/FUZZY.1994.343843

[16] Eren H, Fung CC, Evans J. Implementation of the spline method for mobile robot path control. In IEEE Instrumentation and Measurement Technology Conference. 1999;**2**:739-744. DOI: 10.1109/IMTC.1999.776966

[17] Yamamoto M, Iwamura M, Mohri. Quasi time-optimal motion planning of mobile platforms in the presence of obstacles. In: International Conference on Robotics and Automation. ICRA. pp. 2958-2963. 1999. DOI: 10.1109/ROBOT.1999.774046

[18] Connors J, Elkaim G. Analysis of a Spline Based, Obstacle Avoding Path Planning Algorithm. IEEE Vehicle Technology Conference 2007. DOI: 10.1109/VETECS.2007.528

[19] Connors, Elkaim G. Experimental results of spline based obstacle avoidance of an off–road ground vehicle. ION Global Navigation Satellite Systems Conference. 2007. DOI: 10.1.1.154.2012

[20] Berglund T, Jonsson H, Sderkvist I. An obstacle-avoiding minimum variation B-spline problem. In: International Conference on Geometric Modeling and Graphics. 2003. DOI: 10.1109/GMAG.2003.1219681

[21] Shiller Z, Gwo YR. Dynamic motion planning of autonomous vehicles. IEEE Transactions on Robotics and Automation. 1991;**7**:241. DOI: 10.1109/70.75906

[22] Piegl L. On NURBS: A survey. In IEEE Computer Graphics and Applications. 1991;**11**(1): 55-71. DOI: 10.1109/38.67702

[23] Tatematsu N, Ohnishi K. Tracking motion of mobile robot for moving target using NURBS curve. In Int. Conference on Industrial Technology. 2003;**1**:245-249. DOI: 10.1109/ICIT.2003.1290283

[24] Aleotti J, Caselli S. Trajectory clustering and stochastic approximation for robot programming by demonstration. In: IEEE/RSJ Intelligent Robots and Systems. pp. 1029-1034. 2005. DOI: 10.1109/IROS.2005.1545365

[25] Aleotti J, Caselli S. Trajectory reconstruction with NURBS curves for robot programming by demonstration. In: IEEE International Symposium on Computational Intelligence in Robotics and Automation. pp 73-78. 2005. DOI: 10.1109/CIRA.2005.1554257

[26] Aleotti J, Caselli S. Grasp recognition in virtual reality for robot pregrasp planning by demonstration. In: IEEE International Conference on Robotics and Automation, ICRA. pp. 2801-2806. 2006. DOI: 10.1109/ROBOT.2006.1642125

[27] Montés N, Mora MC, Tornero J. Trajectory generation based on rational Bézier curves as clothoids. In: IEEE Intelligent Vehicles Symp., Istambul, Turkey. 2007;**505**:505-510. DOI: 10.1109/IVS.2007.4290165

[28] Montés N, Herraez A, Armesto L, Tornero J. Real-time clothoid approximation by rational bézier curves. In: International Conference on Robotics and Automation. Pasadena, CA, USA: ICRA. pp 2246-2251. 2008. DOI: 10.1109/ROBOT.2008.4543548

[29] Jaouni H, Khatib M, Laumond JP. Elastic bands for nonholonomic car-like robots: Algorithms and combinatorial issues. In: 3rd International Workshop on the Algorithmic Foundantions of Robotics (WAFR'98). 1998. ISBN:1-56881-081-4

[30] Khatib M, Jaouni H, Chatila R, Laumond JP. Dynamic path modification for car–like nonholonomic mobile robots. IEEE International Conference on Robotics and Automation, ICRA. 1997;**4**:2920-2925. DOI: 10.1109/ROBOT.1997.606730

[31] Graf B, Hostalet JM, Schaeffer C. Flexible path planning for nonholonomic mobile robots. In: 4th European workshop on advanced mobile robots (EUROBOT 01). pp. 199-206, Lund, Sweden, 2001. ISBN: 91-631-1464-X

[32] Nagatani K, Iwai Y, Tanaka Y. Sensor based navigation for car–like mobile robots using generalized voroni graph. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2. 2001, pp 1017-1022. DOI: 10.1109/IROS.2001.976302

[33] Hwang JH, Arkin RC, Know DS. Mobile robots atyour fingertrip: Bézier curve on–line trajectory generation for supervisory control. IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, vol. 2003;**2**:1444-1449. DOI: 10.1109/IROS.2003.1248847

[34] Skrjanc I, Klancar G. Cooperative collision avoidance between multiple robots based on Bézier curves. In: Int. Conf. Information Technology Interfaces, pp. 451-455. 2007. DOI: 10.1109/ITI.2007.4283813

[35] Fujimori A, Teramoto M, Nikiforunk PN, Gupta MM. Cooperative collision avoidance between multiple robots. Journal of Robotic Systems. 2000;**17**(7):347-363. DOI: 10.1002/1097-4563(200007)17:7<347::AID-ROB1>3.0.CO;2-A

[36] Lamiraux F, Bonnafous D, Lefebvre O. Reactive path deformation for non- holonomic mobile robots. IEEE Transactions on Robotics and Automation. 2004;**20**:967-977. DOI: 10.1109/TRO.2004.829459

[37] Lizarraga M, Elklaim G. Spatially deconflicted path generation for multiple UAVs in a bounded airspace. In: IEEE/ION Position, Location & Navigation Symp. 2008. pp. 633-640. DOI: 10.1109/PLANS.2008.4570041

[38] Choi J. Real-time obstacle avoiding planning for autonomous ground vehicles. PhD thesis. California: University of California. December 2010. UMI Number:3442811

[39] Choi J, Curry R, Elkaim G. Path Planning Based on Bézier Curve for Autonomous Ground Vehicles. vol. 1. 158-166. IEEE Computer Society. 2008. DOI: 10.1109/WCECS.2008.27

[40] Choi J, Curry R, Elkaim G. Collision Free Real-Time Motion Planning for Omnidirectional Vehicles. In: European Control Conference. August 2009. ISBN: 978-3-9524173-9-3

[41] Choi J, Curry R, Elkaim G. Obstacle avoiding real-time trajectory generation and control of omnidirectional vehicles. In: American Control Conference (ACC). pp 5510-5515. St Louis, Missouri. 2009. DOI: 10.1109/ACC.2009.5160683

[42] Choi J, Curry R, Elkaim G. Smooth path generation based on Bézier curves for autonomous vehicles. In: World Congress on Engineering and Computer Sciences, WCECS. 2009. pp 668-673. DOI: 10.1.1.294.4485

[43] Choi J, Curry R, Elkaim G. Continuous curvature path generation based on Bézier curves for autonomous vehicles. International Journal of Applied Mathematics. 2010;**40**(2). DOI: 10.1.1.294.6438

[44] Choi J, Curry R, Elkaim G. Curvature-continuous trajectory generation with corridor constraint for autonomous ground vehicles. 49th IEEE Conference on Decision and Control, CDC. 2010. DOI: 10.1109/CDC.2010.5718154

[45] Choi J, Curry R, Elkaim G. Real-Time Obstacle-Avoiding Path Planning for Mobile Robots. In: AIAA Guidance, Navigation and Control, AIAA GNC, Toronto. 2010. DOI: 10.2514/6.2010-8411

[46] Choi J, Elkaim G. Bézier curves for trajectory guidance. In: World Congress on Engineering and Computer Sciences. pp. 625-630. San Francisco, CA, 2008. ISBN: 978-988-98671-0-2

[47] Neto A, Macharet DG, Campos MFM. Feasible RRT-based path planning using seventh order Bézier curves. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp 1445-1450. 2010. DOI: 10.1109/IROS.2010.5649145

[48] Hilario L, Montés N, Falcó A, Mora MC. Real-time trajectory deformation for potential fields planning methos. In: IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS. pp 1567-1572. San Francisco, CA, 2011. DOI: 10.1109/IROS.2011.6094560

[49] Wu OB, Xia FH. Shape modification of Bézier curves by constrained optimization. Journal of Zhejiang University Science. pp. 124-127. 2005. ISSN: 1009-3095

[50] Mora MC, Tornero J. Path planning and trajectory generation using multi-rate predictive artificial potential fields. Proc. IEEE/RSJ IROS. 2008:2990-2995. DOI: 10.1109/IROS.2008.4651091

[51] Mora MC, Tornero J. Predictive and multirate sensor-based planning under uncertainty. IEEE Trans. on Intelligent Transportation Systems. 2015;**16**(3):1493-1504. DOI: 10.1109/TITS.2014.2366974

Chapter 8

Motion Planning for Mobile Robots

Xiangrong Xu, Yang Yang and Siyu Pan

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.76895

Abstract

This chapter introduces two kinds of motion path planning algorithms for mobile robots or unmanned ground vehicles (UGV). First, we present an approach of trajectory planning for UGV or mobile robot under the existence of moving obstacles by using improved artificial potential field method. Then, we propose an I-RRT* algorithm for motion planning, which combines the environment with obstacle constraints, vehicle constraints, and kinematic constraints. All the simulation results and the experiments show that two kinds of algorithm are effective for practical use.

Keywords: path planning, trajectory planning, mobile robots, unmanned ground vehicles (UGV)

1. Introduction

In recent years, the intelligent mobile robots have played an important role in industry, agriculture, aerospace, and space exploration, and it becomes a hotspot issue in many research fields. The robot is divided into two categories from the application environment, named industrial robots and special robots. The industrial robot is industrial oriented multi-joint manipulator or multi-degree of freedom robot, while the special robot is opposite to industrial robots, which is used for non-manufacturing environment or service including service robots, underwater robot, entertainment robot, military robots, agricultural robots, and robotic machines. If we want the robot autonomous mobile in configuration space, the very first thing we will do is the path planning, which can be defined as the process of finding a collision-free path for a robot from its initial position to the goal or target point by avoiding collisions with any static obstacles or other agents present in its environment. In order to solve these problems, many domestic and foreign scholars researched and put forward many theories and methods of the path planning [1–4]. The traditional path planning algorithms, such as ant

colony algorithm, genetic algorithm, and artificial potential field algorithm, dealing with some motion planning have its unique superiority. In addition, the algorithm based on potential field or heuristic function, such as A*, D*, and artificial potential field method, in addressing the problem of planning can meet the requirements of real-time and the optimality. This chapter introduces two kinds of path planning algorithm of the robot, including mobile robots and unmanned ground vehicles (UGV).

2. Path planning for robots with existence of moving obstacles

In this chapter, we present an approach of the ground mobile vehicle or robot trajectory planning under the existence of moving obstacles by using improved artificial potential field method. The potential field intensity and strength instead of force vectors was adopted in the planning. Considering the speed effect of mobile obstacles and mobile robot, the velocity information was introduced into potential field function and an "added potential field" was also applied to guide the mobile robot to be free of collision with local obstacles. Based on the new method, all the potential field intensity was considered and summed by algebraic style, then the genetic trust region algorithm was used to search the minimum sum points of potential field intensity within the motion space and scope, which the mobile robot can reach target during a sampling period, and the global optimization trajectory was consisted of all the minimum points. Simulation and experiment results show that better results of path planning for a mobile robot in a complex environment with the existence of moving obstacles can be achieved using this new approach.

The problem of mobile robot path planning under a dynamic environment in mobile robot domain is hot and difficult. Its task is to find a path, which is from the initial state to goal state. Currently, common path planning methods are artificial potential field method, grid method, neural network method, chaos genetic algorithm, aand so on [5, 6]. Artificial potential field method in the algorithm of path planning is relatively mature and efficient, because of the simplicity of the mathematical calculation, it is widely used. However, there are the issues of local minimum point and destination unreachable in traditional artificial potential field method. There are a variety of ways to escape from the local minimum point, such as random fleeing method, heuristic search, walking along the wall, Tangent bug's method, and so on [7, 8], These methods need to apply an additional control force to the robot, which could not fundamentally solve problems. An improved artificial potential field method based on genetic algorithm was proposed in [8] to solve the problem of goal unreachable and local minimum, but the convergent speed of the algorithm still needs to be improved.

In this chapter, aimed at the shortcoming of traditional artificial potential field method, an improved artificial potential field applied in a dynamic environment is proposed. Firstly, an improved artificial potential field model is established; then the genetic trust region algorithm is applied to solve the sub-target point problem of the improved artificial potential field model and then the optimal path is planned.

2.1. Artificial potential field model

2.1.1. The shortcoming of traditional potential field model

There are some problems when this method is applied to robot path planning:

1. Because the traditional artificial potential field method applies virtual force to control the movement of the robot, it is possible for a robot that it cannot go through a narrow passage when two obstacles are near to each other. Moreover, if robot, obstacles and target point are on the same straight line, the robot controlled by force can only move on the straight line repeatedly, but it cannot reach to the target point.
2. If obstacles are near to target point, it will be possible that repulsive force is greater than attractive force, leading to the problem of goal unreachable (GNRON).
3. The robot has not yet arrived at target point nearly, however, the summation of suffered forces is zero, consequently, it will fall in local minimum point and stop moving.

2.1.2. Improved measures

Aimed at the defects of traditional artificial potential field model, improved measures are proposed as below:

1. Turning the attractive force of target to the robot and the repulsive force of obstacles to the robot into a kind of potential field intensity, a method of calculating potential field is applied to replace traditional vector force control.
2. Adding a coefficient entry $\|X - X_\mu\|_2$ in the gravitational potential of obstacles, when the robot is close to the target point, gravitational potential is reducing as well as repulsion potential. Finally, they are zero until the robot arrives at the target point, so the problem of obstacles and target point being too close causing goal unreachable.
3. For a "deadlock" issue caused by local minimum point, the "added potential field" is introduced to guide the robot to walk out the local minimum point, that is, adding an extra potential field U_{add}.

2.1.3. Improved potential model

According to above improved measures, the improved artificial potential field model is proposed. The attractive force model of the target to a full range of vehicle's body is:

$$U_{att}(X) = 0.5k\rho^2(X, X_g) \tag{1}$$

where $\rho(X, X_g)$ is the distance between the current location of the central point of mobile vehicle's body and target point; k is a proportional gain coefficient; X is the position $[x, y]^T$ of robot's central point in movement space; and X_g is the target point position $\left[x_g, y_g\right]^T$.

Repulsive potential model of the i-th static obstacles on the full range of movement of the body is:

$$U_{reps}(X_i) = \begin{cases} 0.5\eta\left(\frac{1}{\rho(X,X_i)} - \frac{1}{\rho_0}\right)^2 \left\|X - X_g\right\|_2, \\ \qquad \text{if } \rho(X,Xi) \le \rho o \\ 0, \\ \qquad \text{if } \rho(X,Xi) > \rho o \end{cases}$$

where $i \in (1, 2''', n)$, n is the summation of static obstacles; $\rho(X, X_i)$ is the shortest distance of between current location of the center of mobile vehicle's body and the i-th obstacle; ρ_0 the effective effect distance of obstacle; and η is proportional position gain coefficient.

Dynamic obstacles are in motion, and it cannot reflect environment information completely if only taking location into consideration. Bring the relative speed of dynamic obstacles and robots into potential field function, Repulsive potential model of the i-th static obstacles on the full range of movement of the body is got:

$$U_{reps}(X_i) = \begin{cases} 0.5\eta\left(\frac{1}{\rho(X,X_i)} - \frac{1}{\rho_0}\right)^2 \begin{pmatrix} \left\|X - X_g\right\|_2 + \\ \zeta \left|v - v_r \sin(\varphi - \Phi)\right| \end{pmatrix}, \\ \qquad \text{if } \rho(X,Xi) \le \rho o \\ 0, \\ \qquad \text{if } \rho(X,Xi) > \rho o \end{cases}$$

where $r \in (1, 2, \ldots, m)$, m is the summation of mobile obstacles; ζ is proportional coefficient; V is the current speed of the mobile body, $v \in (2v_{\max}/3, v_{\max})$; $v_{\max} t_0$ is the current speed of the r-th dynamic obstacles; ϕ is the current movement direction of the mobile body; and φ is the current movement direction of the r-th dynamic obstacle.

When the robot is in local minimum point, the "added potential" is brought to figure the problem of local minimum out, the added potential model is:

$$U_{add}(X) = \begin{cases} s\rho^2(X, X_g), \rho(X, X_g) > \rho_a; \\ 0, \rho(X, X_g) \le \rho_a \end{cases}$$

where ρ_a judgment distance of whether the mobile body reaches to a target point; s is a proportional coefficient.

Therefore, the whole potential field intensity of a range of mobile body is shown in (2). When the robot is in local minimum point, taking (2) plus the added potential as the total potential field value (as shown in (2)).

$$U = U_{att}(X) + \sum_{i=1}^{n} U_{reps}(X_i) + \sum_{r=1}^{m} U_{repm}(X_r). \tag{2}$$

$$U = U_{att}(X) + \sum_{i=1}^{n} U_{reps}(X_i) + \sum_{r=1}^{m} U_{repm}(X_r) + U_{add}(X). \tag{3}$$

Based on above model, every sampling period all regard the minimum point of potential field sum as a sub-goal point, and multiple sub-target is consisted of global optimization path. The sum of potential field is shown in (2). In order to avoid the case of target vibrating in the vicinity of local minimum point, vector synthesis method is used to judge whether the robot is in local minimum point, if in the local minimum point, potential field sum contains added potential, as shown in (3). Assuming the maximum speed of robot is $v_{\max}$, the sampling period is t_0, so when robot in the reachable range of every sampling period, it takes current location as a center, and $v_{\max}t_0$ is a radius of the circle. The speed of robot movement should not too big or too small in order to ensure the stability of robot movement and performing efficiency, so a sub-goal point can be chosen from an annular region, $R \in (2V_{\max}t_0/3, V_{\max}t_0)$, $\theta \in (0, 2\pi)$.

As **Figure 1** shows, the shaded part of the annular region is the optional region of the sub-target point. Therefore, solving the sub-target issue is the key to improve artificial potential field method.

2.2. Solving target point based on genetic trust region

In order to improve the efficiency of path planning, below two situations will be considered:

(1) When the robots move in the free space far away from obstacle ($\leq \rho_0$), that solving the minimum point of the sum of annular region potential field intensity shown in **Figure 1**. The trust region method in optimal search algorithm has good reliability and fast convergence, which offers a new idea to solve the sub-target. However, its iteration speed sometimes is affected by the radial trust. When the iteration speed is affected by the radius of trust region, using genetic algorithms to solve a point, which is better than one in the current iteration point, then, based on the point the trust region is restarted until the optimized point is found, which can greatly improve the convergence speed of algorithm [9–11].

The points of the shaded part can show as $x' = x + R\cos\theta, y' = y + R\sin\theta$. Therefore, Eqs. (2) and (3) are the function about variables R and θ, $R \in (2V_{\max}t_0/3, V_{\max}t_0)$, $\theta \in (0, 2^*3.14)$. assuming $U_{add}(z)z_1' = x + R\cos\theta, z_2' = y + R\sin\theta, z = (z_1, z_2)$. Using genetic trust region algorithm to solve the target function of sub-goal point as (4) shows, in other words, that solving a class of linear constrained optimized problem, when that the robot is located in local minimum point, (4) and added potential $U_{add}(z)$ are regarded as the target function.

$$\min U(z) = U_{att}(z) + \sum_{i=1}^{n} U_{reps}(z_i) + \sum_{r=1}^{n} U_{repm}(z_r) \tag{4}$$

Using quadratic approximation and constructing constrained trust region sub-problem:

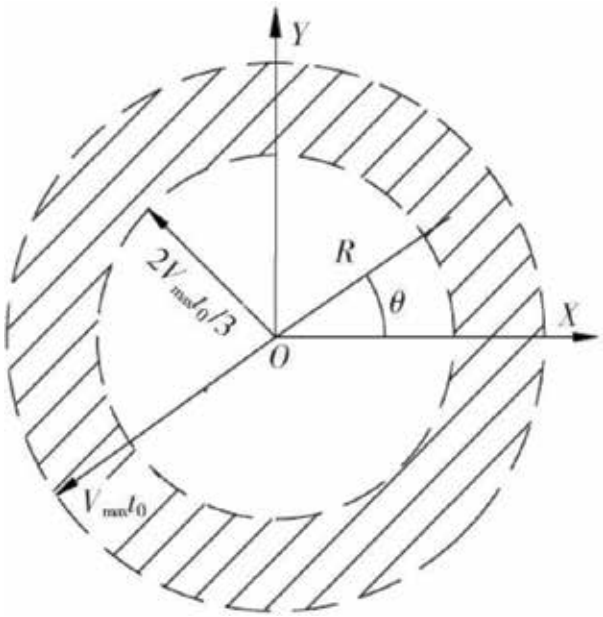

Figure 1. The optional range of sub-target point.

$$\begin{aligned} &\min q_k(d) = g_k^T d + 0.5 d^T G_k d \\ &s.t. \|d\|_2 \le \Delta_k, z_k + d_k \in \Omega \end{aligned} \tag{5}$$

where $g_k = \nabla U(z_k)$; Δ_k is the radius of trust region; $G_k = \nabla^2 U(z_k)$. solving G_k is very complicated, and using BFGS formula of quasi-Newton to structure Hessian matrix B_k, which is approximate to G_k. d_k is the decline tentative step. Ω is the range of R and θ.

The algorithm contains, and remark shows: $z \in R^2$, $z_k = (z_{k1}, z_{k2})$, $z_{k+1} = \left(z_{(k+1)1}, z_{(k+1)2}\right)$, $z^e{}_{k+1} = \left(z^e{}_{(k+1)1}, z^e{}_{(k+1)2}\right)$, $y_k = g_{k+1} - g_k$, := value assignment, $B_{k+1}^{BFGS} = B_k + y_k y_k^T / y_k^T d_k - B_k d_k B_k^T / d_k^T B_k d_k$, the amount of actual decline in the ratio of the amount of pre-estimated decline is:

$$r_k = \Delta U_k / \Delta q_k = \frac{U(z_k) - U(z_k + d_k)}{q_k(0) - q_k(d_k)}.$$

The first step in the algorithm.

Step 1: Initialization, given: z_0, $\Delta_0 > 0$, $\varepsilon_1 > 0$, $\varepsilon_2 > 0$, $\varepsilon_3 > 0$, $a > 0$, $b > 0 M > 1$.

$$0 < \eta_1 < \eta_2 < 1, B_0 =: I_{2x2}, 0 < \beta_1 < \beta_2 < 1 \le \beta_3, k := 0$$

Step 2: Calculating g_k, if $\left\|g_k\right\|_2 < \varepsilon_1$, stop calculating, and output the result.

Step 3: Solving sub-problem in Eq. (5), and get the decline tentative step d_1.

Step 4: Calculating r_k, if $r_k > \eta_1$ meanwhile $z_k + d_k \in \Omega$, so, $z_{k+1} := z_k + d_k$ rectifies B_{k+1}, or $z_{k+1} := z_k$, $B_{k+1} := B_k$.

Step 5: Choosing Δ_{k+1}, let it meet:

$$\Delta_{k+1}\begin{cases}\{[\Delta_k, \beta_3\Delta_k], r_k > \eta_2;\\ [\beta_2\Delta_k, \Delta_k], r_k \in [\eta_1, \eta_2];\\ [\beta_1\Delta_k, \beta_2\Delta_k], r_k < \eta_1.\end{cases}$$

Step 6: If $\Delta_{k+1} < \varepsilon_2$, $\|U_{k+1} - U_k\| < \varepsilon_3$, and using genetic algorithm to quickly solve min $U(z_{k+1}^e)$ so that an iteration point which is better than the current point, go to step 7, Or back to step 8.

Step 7: If $\|U_{k+1} - U_k\| > M\|U_{k+1} - U_k\|$, so order $z_{k+1} := z_{k+1}^e G_{k+1} := \Delta_0$, back to step 2; or back to step 8.

Step 8: Using BFGS formula to modify B_k and gets B_{k+1}, $k := k + 1$, back to step 2.

ε_2, ε_3, M values can be adjusted to control the number of times the call number of sub-issues of a genetic algorithm to achieve the optimization algorithm speed, regulating ε_1 also can regulate the speed of convergence of the algorithm, but there are trade-offs to optimize the value obtained, need to be considered in accordance with the actual situation.

When the convergence speed is influenced by the radius of the trust region in step 6 of algorithm 1, using a genetic algorithm to figure $\min U(z_{k+1}^e)$ out must meet:

$$z_{k+1}^e \in \Omega, c < z_{k+1}^2, l < z_{k+1}^e < n.$$

where

$$c = z_{k1}^e - a, f = z_{k1}^e + a, l = z_{k2}^e - b, n = z_{k2}^e + b,$$

when solving $z_{(k+1)1}^e$, $z_{(k+1)2}^e$, c, f, l, n are all known quantity.

Fitness function: establishing the mapping relationship between the objective function and the function of moderate: $G(z) = 0.618^{U(z)}$; adopting the binary code to encode; replication strategy to preserve the best individual mixed roulette selection; crossover operator is single point crossover; and mutation operator for the basic bit mutation.

The procedure of algorithm 2:

Step 1: Parameter initialization: population size N, crossover probability P_c, mutation probability P_m, current algebraic $T_c = 0$, maximum algebraic $T_{\max}$, and $0 < k < 0.618$. The coding initial solution produces a unit corresponding to the initial solution. And it randomly produces two l-bit binary string working as one unit, and it does not stop until it generates N units.

Step 2: Solving adaption value of every individual in the current group, to get the optimization z_{1b}, z_{2b}: described as:

If

$$T_c \le T_{\max}$$

If

$$G(z_b) - G(z_k) > k \text{ and } z \in \Omega, \text{ then } z^e_{(k+1)1} \coloneqq z_{1b},$$

$z^e_{(k+1)2} \coloneqq z_{2b}$, stopping calculating, and backing to the step1 of algorithm 1.

ELSE backing to step 3.

ELSE updating initial parameters, baking to step 1.

Step 3: Replacing the two worst individuals in the two optimizations of the current generation, then selecting N individuals by using roulette selection.

Step 4: Manipulating cross mutation to produces a new population. Backing to step 2.

2.3. Simulation and experiment

We select the initial parameters of the algorithm:

$\triangle_0 = 0.05\|U(z_0)\|, \varepsilon_1 = 0.1, \varepsilon_2 = 0.03, \varepsilon_3 = 0.05, a = b = 0.5, M = 1.5, \eta_1 = 0.15, \eta_2 = 0.3, B_0 = I_{2x2}$, $\beta_1 = 0.35, \beta_2 = 0.75, \beta_3 = 1.25, .$

$N = 20, P_c = 0.99, P_m = 0.05, T_{\max} = 100, k = 0.1,$ coding length$l = 32. v_{\max} = 0.3m/s, t_0 = 3s,$ $z_0 = (5v_{\max}t_0/6.0), k = 1, \eta = 2, \zeta = 0.1, n = 4, m = 1, .$

$s = 0.3$. The speed of mobile obstacles.

$v_r = 0.2(0.5) + r \; and \; (m)/s,$

$$\varphi = \pi/2, \rho_0 = 2m, \rho_a = 0.15m$$

The original point of robots is (0, 0), the target point is (20, 24), and the unit is m (meter).

The simulation results are shown in **Figures 4** and **5**. It can be seen from simulation, that to combine trust region algorithm and the improved artificial potential field method can optimize mobile robot path planning, and get a better solution to local minima and the unreachable target problem. Under the same conditions, a pure genetic algorithm for solving the sub-goal problem, the average convergence time comes to the second level, and the individual points solvers need more than 30 s, therefore, the proposed algorithm is much fast.

Robot trajectory simulation resulting potential field on a typical point 1 seen from **Table 1**, the optimal path in the intensity values obtained are shown in table potential field strength decreases rapidly, and the strong field in the times to reach the target point, the basic value to 30 s.

To verify the validity of the path planning algorithm, an experiment was made based on the current conditions as shown in **Figures 2** and **3**. The equipment used in the experiment include mobile robot, laser tracker sensor, distance detector sensor, digital compass, dynamics obstacles, and still obstacles.

http://dx.doi.org/10.5772/intechopen.76895

Point	x/m	y/m	strength of field
1	0. 63	0. 64	460. 440
2	2. 33	2. 74	382. 110
3	4. 48	5. 60	313. 620
4	4. 43	9. 00	273. 710
5	6. 89	11. 61	216. 690
6	9. 54	14. 04	142. 790
7	10. 42	17. 26	108. 600
8	13. 30	19. 41	72. 980
9	16. 27	21. 41	38. 310
10	19. 97	23. 97	0. 001

Table 1. Magnitude of field of the point at the trajectory.

The laser tracker can precisely measure the center point of the coordinate (x, y) when mobile robot is in motion. Then it can send the distance information to the robot controller through a local network and get the location of the robot in the space. The mobile robot is shaped like a cylinder, with 0.6 m in diameter, 12 ultrasonic distance detect sensors with 0.1–6.0 m detect distance, and 32 infrared distance sensors with 0.1–0.8 m detect distance. The angle between two adjacent sensors is 30°. The measurement time is 0.015 s. The initial position of the robot is (1.375, −2.328), and the target position is (1.248, 1.353), the velocity of moving obstacle is V_r = 0.08 *m/s*, and $\varphi = \pi/6$. The velocity of the mobile robot is V = 0.05 *m/s*, the moving direction can be measured in real-time by the digital compass.

The initial parameters of the experiment are k = 1, n = 2, m = 1 $\eta = 2, \zeta = 0.1, \rho_0 = 1m$, $\rho_a = 0.15m$. The laser tracker recorded many points of the mobile robot, and the approximated trajectory can be obtained from these points as shown in **Figures 4** and **5**. And the relationship between the field strength and algorithm implementation time was listed in **Table 2**. From the experiment, we can see that the algorithm has some advantages compared with those traditional methods. It is faster in implementation and computation.

2.4. Commits

We presented an approach of mobile robot trajectory planning under the existence of moving obstacles by using improved artificial potential field method. To further verify the improved artificial potential field method for moving machines effectiveness of robots path planning, the use of the existing conditions, design path planning experiment = experimental scene shown in "laboratory equipment including full orientation of the mobile robot laser tracker digital

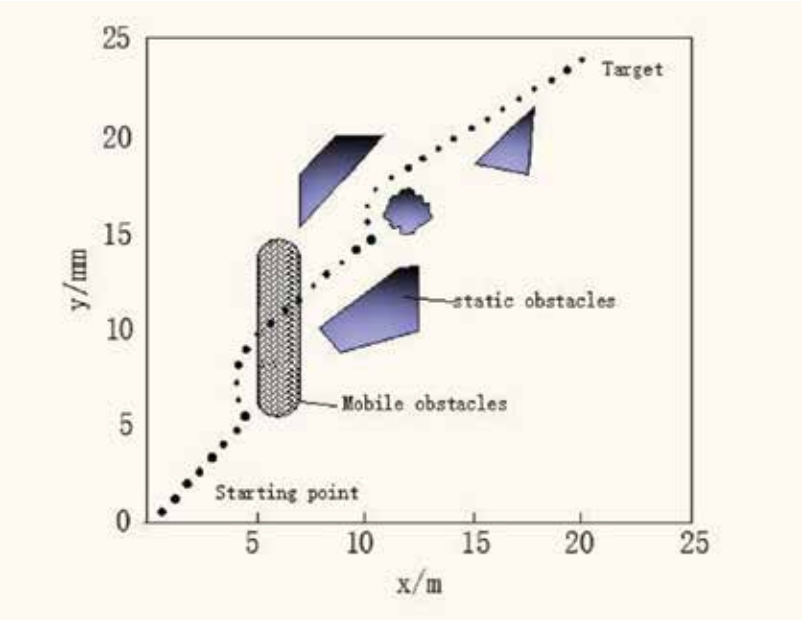

Figure 2. The simulation result of improved artificial potential field method.

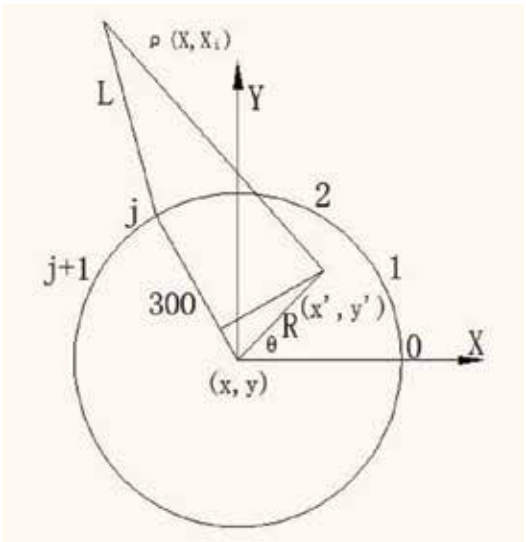

Figure 3. Geometric representation for computing target potion.

distance sensor compass static obstacles and dynamic obstacles. Laser tracker can accurately measure mobile robot motion moving the coordinate value of the center point of the process (x, y), through a wireless local area network location information measured in real-time sent to the robot controller, robot positioning cylindrical omnidirectional mobile robot structure with a diameter 0.6 m, and surrounded by 12 uniform ultrasonic ranging.

http://dx.doi.org/10.5772/intechopen.76895

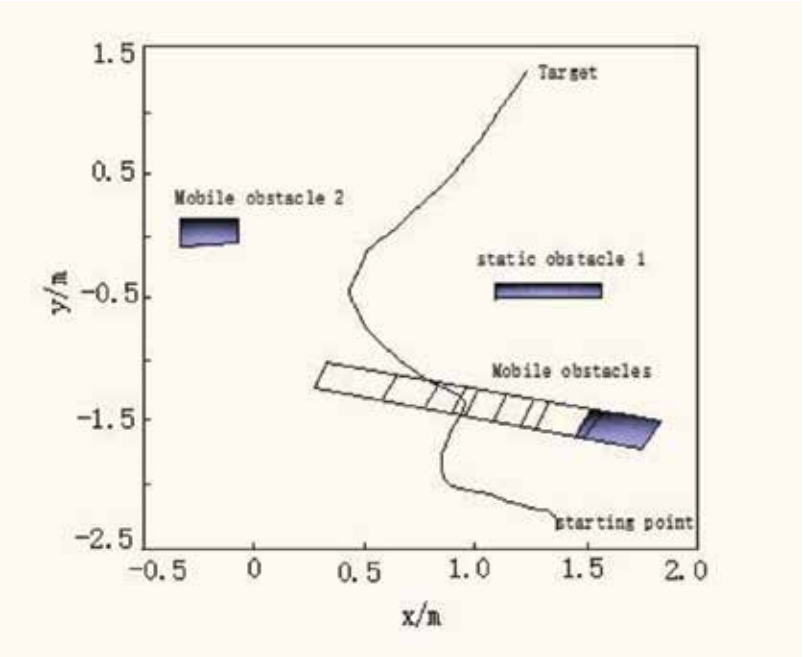

Figure 4. Trajectory of the mobile robot.

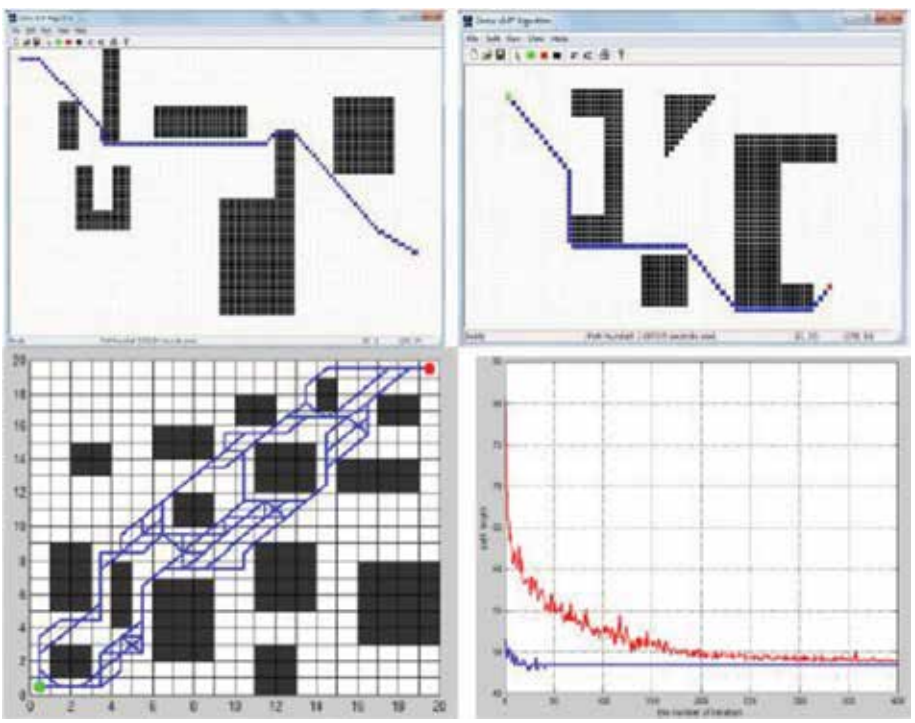

Figure 5. Trajectory planning and simulation (1–3).

Point	x/m	y/m	t/ms	Field strength
1	1.375	−2.328	13	6.783
2	0.845	−1.795	11	5.439
3	0.947	−1.330	15	4.247
4	0.694	−1.075	12	3.703
5	0.475	−0.612	19	2.831
6	0.504	−0.145	14	1.932
7	0.686	0.118	9	1.241
8	0.822	0.363	21	0.981
9	1.031	0.807	12	0.273
10	1.217	1.319	10	0.001

Table 2. Algorithm implementation time and field strength.

3. Improved RRT* motion planning algorithm for mobile robot

Some path planning algorithms such as ant colony algorithm, genetic algorithm, artificial potential field algorithm, and dealing with some planning have its unique superiority. However, in the complex environment and high-dimensional space, the complex of the algorithm will increase sharply, which lead the convergence time too long to solve the problem. Because the algorithm based on potential field or heuristic function does not consider the kinematic and dynamic constraints, its result cannot be executed by the real case [12–17].

In order to solve the problem of high-dimensional path planning, the sampling algorithms have been introduced [18]. Compared with another advanced algorithm, the main advantage of the algorithm based on sampling is avoiding constructing the explicit configuration space, and it has been shown to be an effective solution to the path planning [19]. RRT algorithm based on sampling with rapid convergence rate can be used in unknown obstacle environment [20]. However, there are still some shortcomings in RRT algorithm [21–23]:

1. Its convergence rate is very slow in achieving the optimal solution;
2. Its memory requirements are significantly large due to a large number of iterations used to calculate the optimal path;
3. The rejection of samples, which might not be connectable directly with the existing nodes in the tree, but may lie closer to the goal region and hence could aid the algorithm in determining an optimal path much more rapid.

The domestic and foreign scholars have been studied on these deficiencies and proposed various RRT algorithms to adapt to different application scenarios. In order to improve the efficiency of the expansion of the node, Kuffner and La Valle proposed the RRT-connect [24]. Karaman and Frazzoli first proposed RRT* algorithm, which convergence to the optimal solution [25]. A.H. Qureshi and S. Mumtaz proposed the TG-RRT*, using triangular geometry to select node, in order to reduce the number of iterations required for the optimal solution, so as to make the algorithm rapid convergence [26]. C. Wouter Bac and Tim Roorda proposed by

RRT algorithm on ROS platform for mobile robot path planning research and applied to the motion planning problem in the dense obstacles environment [27].

3.1. Nonholonomic constraints for mobile robot

The integrity constraints include a system of generalized coordinates derivative and integral constraints. The mobile robot is a typical nonholonomic constraint system. The mobile robot's simplified moving model is shown in the **Figure 6**.

Mobile robot's state variables in configuration space (x, y, θ, v, ϕ), let (x, y) represent the center of mobile robot rear axle in the system coordinates, θ represent the angle between the forward direction of the mobile robot and the x-axis, φ represent the angle between the forward direction of the mobile robot and the front wheel direction, v as the speed of the mobile robot, due to the nonholonomic constraints, the wheel is point contact with the ground, and only pure roll no relative sliding at contact.

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = {}^{v\tan\phi}/_{L} \\ \dot{v} = u_0 \\ \dot{\phi} = u_1 \end{cases} \tag{4.1}$$

And the constraint equations:

$$\begin{pmatrix} {}^{dx}/_{dt} \\ {}^{dy}/_{dt} \\ {}^{d\theta}/_{dt} \\ {}^{dv}/_{dt} \\ {}^{d\phi}/_{dt} \end{pmatrix} = \begin{pmatrix} v\cos\theta \\ v\sin\theta \\ v\tan{}^{\phi}/_{L} \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} u_0 + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} u_1 \tag{4.2}$$

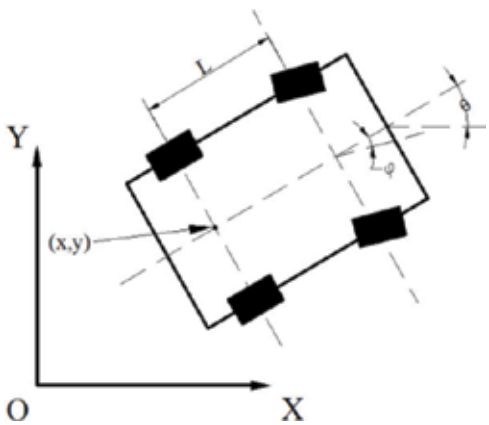

Figure 6. Geometrics of the mobile robot.

u_0 (acceleration) and u_1 (angular acceleration) of the mobile robot control variables. The minimum turning radius (maximum curvature) of the mobile robot is:

$$K_{\max} = \frac{\tan \phi_{\max}}{L} = \frac{1}{R_{\min}} \tag{4.3}$$

Therefore, it is unreasonable without considering the nonholonomic constraints of the robot planning algorithm.

3.2. Description and implementation of the I-RRT*

I-RRT* algorithm showed in **Table 3** is specifically designed for motion planning in complex, cluttered environments where exploration of configuration space is difficult. Let the sets of near vertices from the tree T_a and T_b be denoted by X^a_{near} and X^b_{near}, respectively. The path is connecting P^a_{init} and P_{rand} is denoted by $\sigma_a{}' : [0, s_a]$, while the path is connecting P^b_{init} and P_{rand} is denoted by $\sigma_b{}' : [0, s_b]$.

It starts by picking a random sample P_{rand} from the obstacle-free configuration space X_{free} that is, $x_{rand} \in X_{free}$. It then populates the set of near vertices, X^b_{near} for both trees using the *NearVertices* procedure. It should be noted that a ball region centered at P_{rand} of radius r is formed, and the sets of the near vertices from both trees are computed that is:

---

1. $V_a \leftarrow \{x^a_{init}\}; E_a \leftarrow \phi; T_a \leftarrow (V_a, E_a);$
2. $V_b \leftarrow \{x^b_{init}\}; E_b \leftarrow \phi; T_b \leftarrow (V_b, E_b);$
3. $\sigma_f \leftarrow \infty; E \leftarrow \phi;$
4. $Connection \leftarrow True$
5. $for \quad i \leftarrow 0 \quad to \quad N \quad do$
6. $x_{rand} \leftarrow Sample(i)$
7. $\{X^a_{near}, X^b_{near}\} \leftarrow NearVertices(x_{rand}, T_a, T_b)$
8. $if \quad X^a_{near} = \varphi \quad \&\& \quad X^b_{near} = \varphi \quad then$
9. $\{X^a_{near}, X^b_{near}\} \leftarrow NearestVertex(x_{rand}, T_a, T_b)$
10. $Connection \leftarrow False$
11. $L^a_s \leftarrow GetSortedList(x_{rand}, X^a_{near})$
12. $L^b_s \leftarrow GetSortedList(x_{rand}, X^b_{near})$
13. $\{x_{\min}, flag, \sigma_f\} \leftarrow GetBestTreeParent(L^a_s, L^b_s, Connection)$
14. $if(flag)then$
15. $T_a \leftarrow InsertVertex(x_{rand}, x_{\min}, T_a)$
16. $T_a \leftarrow RewireVertex(x_{rand}, x_{\min}, T_a)$
17. $else$
18. $T_b \leftarrow InsertVertex(x_{rand}, x_{\min}, T_b)$
19. $T_b \leftarrow RewireVertex(x_{rand}, x_{\min}, T_b)$
20. $E \leftarrow E_a \cup E_b$
21. $V \leftarrow V_a \cup V_b$
22. $return \quad (\{T_a, T_b\} = V, E)$

Table 3. The algorithm of I-RRT*.

$$X_{near}^{a} := \left\{ v \leftarrow V_a : v \in \mathrm{B}_{x_{rand}, r} \right\} \tag{4.4}$$

$$X_{near}^{b} := \left\{ v \leftarrow V_b : v \in \mathrm{B}_{x_{rand}, r} \right\} \tag{4.5}$$

In the case of both sets of near vertices being found empty, these sets are filled with the closest vertex from their respective trees instead. The procedure Best Selected Tree returns the nearest vertex on the Best Selected Tree, which is eligible to become the parent of the random sample.

Hence, unlike the connect heuristic, the I-RRT* is not greedy since the connection is only made inside the ball region. Finally, the tree connection generates the end-to-end global path.

3.3. Experimental results

3.3.1. The algorithm simulation

This section presents simulations performed on a 2.4 GHz Intel Core i5 processor with 4 GB RAM. Here, performance results of our I-RRT* algorithm are compared with RRT* and B-RRT*. For proper comparison, experimental conditions and size of the configuration space were kept constant for all algorithms. Since randomized sampling-based algorithms exhibit large variations in results, the algorithms were run up to 50 times with different seed values for each type of environment. Maximum, minimum, and an average number of iterations $\boldsymbol{i}$ as well as time $\boldsymbol{t}$ utilized by each algorithm to reach the optimal path solution is presented in **Table 4**.

In the second environment simulation, chooses some representative to verify the algorithm performance.

Figures 7–9 are three kinds of the optimization algorithm for the same problems. And we can see the I-RRT* is better than the others.

In the third environment model simulation test, using the three different kinds of obstacle environment simulation experiment, we tested the performance of the algorithm in **Figure 10**.

Environment	ALG	$i_{\min}$	$i_{\max}$	i_{avg}	$t_{\min}$	$t_{\max}$	t_{avg}	C	fail
3D-Random obstacles	RRT*	107,810	110,851	108,965	17.8	19.3	18.8	**	6
	B-RRT*	30,901	38,086	36,128	6.4	8.1	7.4	**	4
	I-RRT*	19,507	22,525	21,290	4.4	5.3	5.1	**	1
3D-Multiple barriers	RRT*	1,430,381	1,440,619	1,437,342	242.1	247.4	244.1	205.6	14
	B-RRT*	495,961	503,240	498,972	102.1	106.5	105.1	205.6	6
	I-RRT*	97,885	112,857	111,139	22.3	27.3	26.2	205.6	3
3D-Narrow passage	RRT*	1,277,376	1,301,698	1,290,674	216.9	221.9	218.5	345.1	9
	B-RRT*	533,276	561,347	551,771	110	117.1	115.9	345.1	3
	I-RRT*	127,363	143,806	134,421	30.3	35.2	31.4	345.1	0

Table 4. Experimental results for computing optimal path solution.

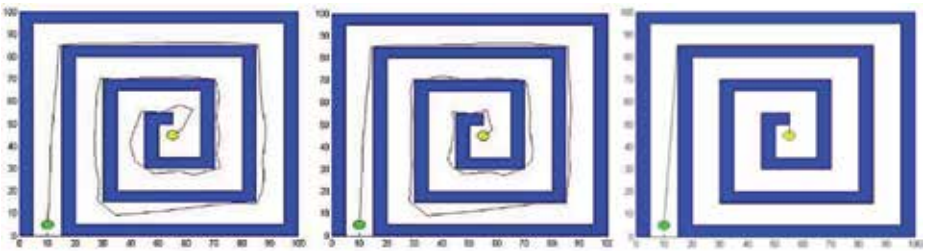

Figure 7. RRT* performance in 2-D environment.

Figure 8. B-RRT* performance in 2-D environment.

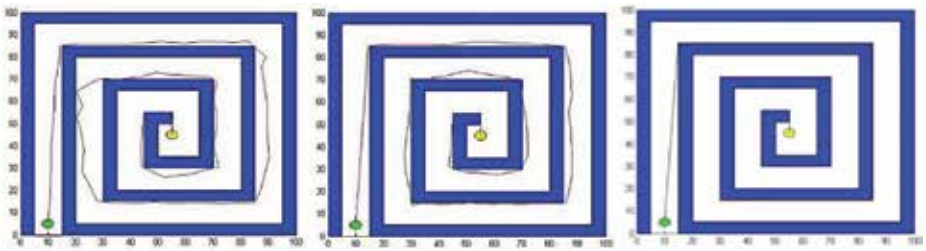

Figure 9. I-RRT* performance in a 2-D environment.

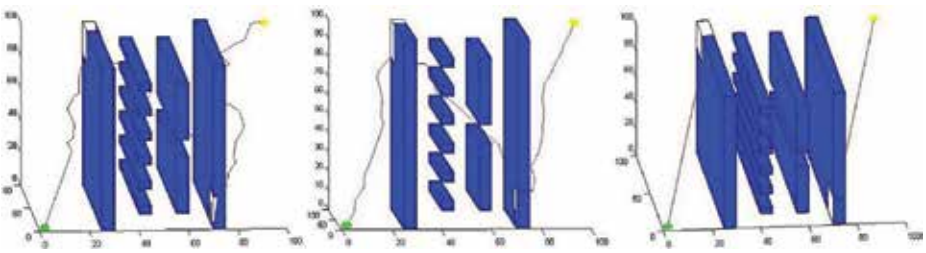

Figure 10. I-RRT* performance in 3-D environment.

To restrain the computational time within reasonable limits, the maximum limit for the number of tree nodes was kept at 3 million.

Although both I-RRT* and B-RRT* were successful in finding the optimal solution, B-RRT* took an extremely large number of iterations to converge in comparison with I-RRT*. B-RRT*

utilizes the partial greedy heuristic approach as discussed earlier, which significantly reduces its ability of convergence to the optimal path solution. **Table 4** shows the convergence from the initial path solution to the optimal path solution by I-RRT* and RRT*, respectively. For determination of the optimal path, the I-RRT* algorithm takes the least number of average iterations (iavg = 111,139) as compared to B-RRT*(iavg = 498,972) and the extraordinarily large number of iterations taken up by RRT*(iavg = 1,437,342).

3.4. Mobile robot simulation experiment on ROS

The simulation of the environment was established under the gazebo. In building a model, a mobile robot called ROS mapping packages build implementation scenario map in **Figures 11–13**.

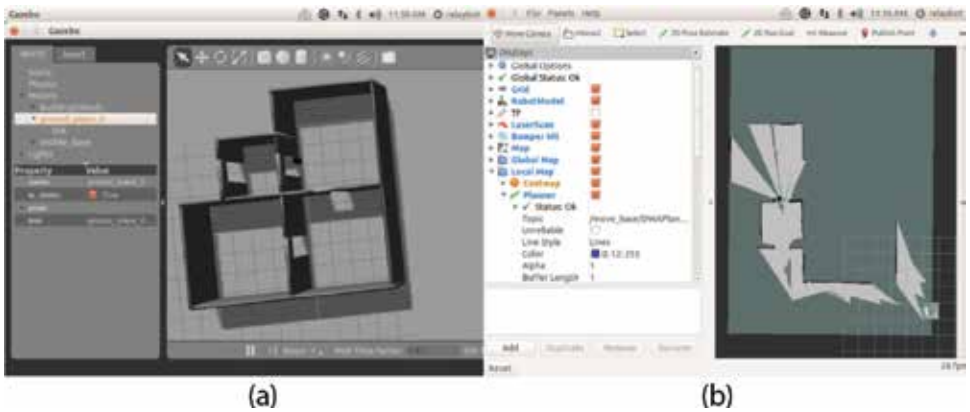

Figure 11. The process of SLAM. (a) SLAM show under gazebo (b) SLAM show under RVIZ.

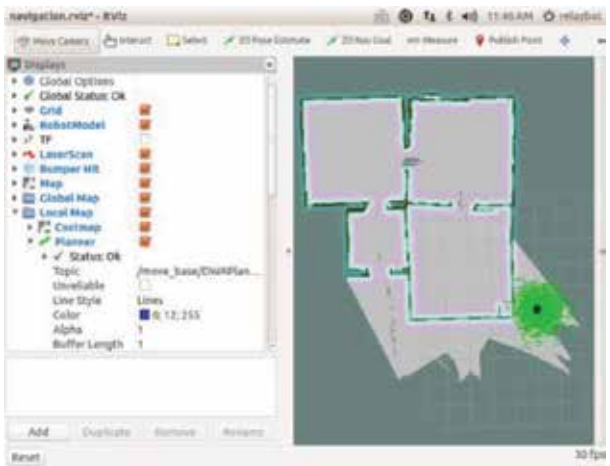

Figure 12. Map information in RVIZ.

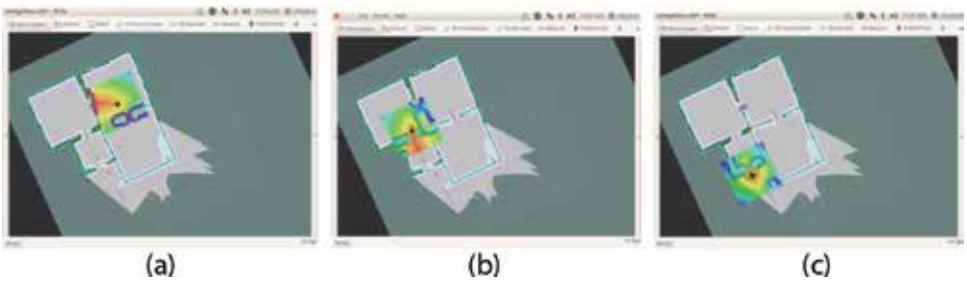

Figure 13. The process of planning. (a) Start planning, (b) the process of planning, and (c) planning complete.

3.5. Static environment experiment

Using the I-RRT* algorithm for mobile robot path planning, we choose a random target on the map and set the position and posture in **Figure 13**. I-RRT* algorithm can generate a barrier-free path, as shown in the green feed, where the trajectory planning algorithm.

Figures show the I-RRT* dynamic planning process, which is real-time planning. I-RRT* algorithm programming device receives the message, if there are new obstacles, in order to avoid obstacles, I-RRT* real-time planning. In this picture perceived color area for mobile robot around obstacles of information, the green line is the initial path planning, the red line for real-time planning path. As a result, the algorithm can be applied to the dynamic environment of the path planning problem.

4. Conclusion

Solving the path planning problems of the ground robot and mobile robot are the premise of more efficient use of the robot. Due to the different types of robots, the problems in the path planning need to be considered, from 2D to 3D. Three path planning algorithms for different types of robots are proposed in this chapter. The problem of path planning for existing industrial robots and mobile robots, including unmanned vehicles and unmanned aerial vehicles, are solved. All the simulations results show that the algorithms proposed in this paper are feasible.

Acknowledgements

This research is supported in part by Natural Science Foundation of China (NSFC) No. 51405001 and 31300125.

Author details

Xiangrong Xu*, Yang Yang and Siyu Pan

*Address all correspondence to: xuxr88@yahoo.com

School of Mechanical Engineering, Anhui University of Technology, Ma'anshan, Anhui, China

References

[1] Fu KS, Gonzales RC, Lee CS. Robotics: Control, Sensing, Vision, and Intelligence. Singapore: McGrawHill. Inc; 1987. ISBN:0070226253

[2] Jung D, Tsiotras P. On-line path generation for small unmanned aerial vehicles using B-spline path templates. AIAA Guidance, Navigation and Control Conference, AIAA. 2008, Vol. 7135. DOI: 10.2514/6.2008–7135

[3] Chandler P, Rasmussen S, Pachter M. UAV cooperative path planning. AIAA Guidance, Navigation, and Control Conference and Exhibit. 2000. pp. 1255-1265. DOI: 10.2514/6.2000-4370

[4] Oscar M, Ulises O, Roberto S. Path planning for mobile robots using bacterial potential field for avoiding static and dynamic obstacles. Expert Systems with Applications. July 2015;**42**(12):5177-5191

[5] Park M, Jeon J, Lee M. Obstacle avoidance for mobile robots using artificial potential field approach with simulated annealing. Proceedings of the 2001 IEEE International Symposium on Industrial Electronics, Pusan. 2001. pp. 1530-1535

[6] Kitamura Y, Tanaka T, Kishino F. 3-D path planning in a dynamic environment using an octree and an artificial potential field. Proceedings of the 1995 IEEE International Conference on Intelligent Robots and Systems. Piscataway: IEEE; 1995. 2474-2481

[7] Janabi S, Vinke D. Integration of the artificial potential field approach with simulated annealing for robot path planning. Proceedings of the 1993 IEEE International Symposium on Intelligent Control Piscataway: IEEE. 1993. 536-541

[8] Hsu CK. Variable structure control design for uncertain dynamic systems with sector nonlinearities. Automatica. 1998;**34**(4):505-508

[9] Ashiru I, Czarnecki C, Routen T. Characteristics of a genetic based approach to path planning for mobile robots. Journal of Network and Computer Applications, 1996;**19**(2):149-169

[10] Chen L, You B. Dynamic robot motion tracking and obstacle avoidance based on artificial potential field method. Control Theory and Application. 2007;**26**(4):8-10

[11] Vadakkepat P, Tan WK. Evolutionary artificial potential fields and their application in real time robot path. Proceedings of the 2000 IEEE Conference on Evolutionary Computation Evolutionary Computation. Piscataway: IEEE. 2000. 256-263

[12] Xu X, Shi DQ, Lu M, et al. Study on mechanical mechanics with a method for minimum-time path planning of robots in cartesian space. Advanced Materials Research. 2013;**703**: 181-185. DOI: 10.4028/www.scientific.net/AMR.703.181

[13] Pellazar MB. Vehicle route planning with constraints using genetic algorithms. Aerospace and Electronics Conference, 1994. NAECON 1994. Proceedings of the IEEE 1994 National. IEEE, 1994: 111-118. DOI: 10.1109/NAECON.1994.333010

[14] Duan H, Yu Y, Zhang X, et al. Three-dimension path planning for UCAV using hybrid meta-heuristic ACO-DE algorithm. Simulation Modelling Practice and Theory. 2010;**18**(8): 1104-1115. DOI: 10.1016/j.simpat.2009.10.006

[15] Wang H, Liu YH, Chen W. Visual tracking of robots in uncalibrated environments. Mechatronics. 2012;**22**(4):390-397. DOI: 10.1016/j.mechatronics.2011.09.006

[16] He J, Hou Z. Ant colony algorithm for traffic signal timing optimization. Advances in Engineering Software. 2012;**43**(1):14-18. DOI: 10.1016/j.advengsoft.2011.09.002

[17] Xu X, Li Y, Yang Y, et al. A method of trajectory planning for Ground Mobile Robot based on ant colony algorithm. IEEE International Conference on Robotics and Biomimetics. IEEE, 2017:2117–2121. DOI: 10.1109/ROBIO.2016.7866642

[18] Elbanhawi M, Simic M. Sampling-based robot motion planning: A review. IEEE Access. 2014;**2**(1):56-77. DOI: 10.1109/ACCESS.2014.2302442

[19] Qureshi AH, Ayaz Y. Intelligent bidirectional rapidly exploring random trees for optimal motion planning in complex cluttered environments. Robotics and Autonomous Systems. 2015;**68**:1-11. DOI: 10.1016/j.robot.2015.02.007

[20] Kavraki L, Latombe JC. Randomized preprocessing of configuration for fast path planning. IEEE International Conference on Robotics and Automation, 1994. Proceedings 3, 2138-2145. IEEE. 1994. DOI: 10.1109/ROBOT.1994.350966

[21] Vonásek V, Saska M, Winkler L, Přeučil L. High-level motion planning for cpg-driven modular robots. Robotics & Autonomous Systems. 2015;**68**:116-128. DOI: 10.1016/j.robot.2015.01.006

[22] Doshi AA, Postula AJ, Fletcher A, Singh SPN. Development of micro-uav with integrated motion planning for open-cut mining surveillance. Microprocessors and Microsystems. 2015;**39**(8):829-835. DOI: 10.1016/j.micpro.2015.07.008

[23] Park KJ, Won M. People tracking and accompanying algorithm for mobile robot using kinect sensor and extended kalman filter. Transactions of the Korean Society of Mechanical Engineers A. 2014;**38**(4):345-354. DOI : 10.3795/KSME-A.2014.38.4.345

[24] Kuffner JJ, Lavalle SM. RRT-connect: An efficient approach to single-query path planning. IEEE International Conference on Robotics and Automation, 2000. Proceedings. ICRA (Vol. 2, pp. 995-1001). 2000. IEEE. DOI: 10.1109/ROBOT.2000.844730

[25] Karaman S, Frazzoli E. Sampling-based algorithms for optimal motion planning. International Journal of Robotics Research. 2011;**30**(7):846-894. DOI: 10.1177/0278364911406761

[26] Qureshi AH, Mumtaz S, Iqbal KF, Ayaz Y. Triangular geometry based optimal motion planning using RRT*-motion planner. IEEE, International Workshop on Advanced Motion Control. 2014:380-385. DOI: 10.1109/AMC.2014.6823312

[27] Bac CW, Roorda T, Reshef R, Berman S, Hemming J, Henten EJV. Analysis of a motion planning problem for sweet-pepper harvesting in a dense obstacle environment. Biosystems Engineering. 2016;**146**:85-97. DOI: 10.1016/j.biosystemseng.2015.07.004

Chapter 9

Design and Implementation of a Demonstrative Palletizer Robot with Navigation for Educational Purposes

Dora-Luz Almanza-Ojeda,
Perla-Lizeth Garza-Barron,
Carlos Rubin Montoro-Sanjose and
Mario-Alberto Ibarra-Manzano

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.72872

Abstract

Nowadays, many kinds of robots are used in industries to help in manufacturing or placing objects. However, teaching young people and children about robot design and work can be difficult, turning this into a complicated area for them. This chapter provides a detailed description of the design and implementation of a robotic arm mounted on a mobile robot using the LEGO Mindstorms NXT kit® and the starter kit DaNI 2.0, designed by National Instruments®. The mobile palletizer robot takes a box from place A and navigates in the indoor environment until it reaches a predefined place B. The characterization of the robotic arm is based on a parallel structure considering that the end-effector has only two points to hold the object; the gripper is also built using LEGO®. The robot performs the path computed using an A-star algorithm; moreover, actions like moving up and down, opening and closing the gripper and picking up the box and putting it down are executed by the robotic arm using the central unit of the NXT kit. Each stage of the robot design and implementation is explained in detail using diagrams and 3D graphical views with the aim of illustrating the implementation step by step for educational purposes (mainly for young people or children).

Keywords: mobile robot, robotic arm, parallel structure, path planning

1. Introduction

Robots have been used in applications such as industry, medicine, agriculture, space, education, underwater exploration and many others. Manufacturing processes in industries have

increased considerably the use of robotic arms to automate repetitive and tedious tasks performed under difficult conditions for workers. Moreover, the use of mobile robots in industries also improves the efficiency and accelerates the production process. Mobile robots are equipped with sensors to analyze and interpret information about the environment during navigation [1]. Some applications of mobile robots in industry are as follows:

1. inspection,
2. production control,
3. transport of different kinds of objects by means of palletizing tasks [2].

The palletizing of objects (essentially boxes) in the industry is the process to accommodate boxes on a pallet that is usually performed by fixed robotic arms [3, 4]. In cases when the destination is not fixed, mobile robots are also used to place boxes to a destination. For instance, magnetic strip-guided robots transport the merchandise successfully, albeit only following a linear path. Therefore, one of the best solutions for palletizing objects from an origin to a destination involves the use of robotic arms mounted on mobile robots.

The palletizing task requires a path planning strategy which consists in finding an obstacle-free path for mobile robot navigation from one place to another. Many path planning strategies can be found in the literature for various applications, ranging from video game programming to outdoor autonomous navigation of robots. Path planning methods are based on simplifying the searching area to a 2D matrix in which each element represents a reduced square area of the navigation area (that will be interpreted as a cell) [5]. Thus, each cell can be navigable or not depending on the obstacles on it, and a resulting path is obtained if a set of adjacent navigable cells from the origin to destination is found.

The aim of implementing a Box Palletizing Robot is to encourage young people to explore robotic issues as modular tasks that require design, mathematical modeling, programming and some interest and creativity. In this context, many robotic kits and prototypes have been introduced by different companies such as Vex [6], Arduino, Lego, Zowi from BQ [7], to name but a few. However, in this chapter, we present a palletizer robot that combines two robotic kits: the mobile robot platform Dani from National Instruments (NI) and the Lego Mindstorms NXT 2.0 8547 model used to build the robotic arm. Both robotic kits require basic, medium and high levels of knowledge in line with the final purposes. In our case, we will describe the design, programming and synchronization of both kits.

Additionally, the path planning strategy used in this project is based on the A-star algorithm and basic strategies to control the robotic arm. The characterization of the robotic arm is based on a parallel structure, and it has been built using the LEGO NXT kit. To improve the compatibility between the robotic arm and the robot mobile, the LEGO NXT is programmed on LabVIEW [8], a trademark software of NI, to use the starter kit, which is a robot also distributed by NI. An Ethernet connection is used for communication between the PC and the mobile robot, while a Bluetooth connection is used for communication with the robotic arm [9].

http://dx.doi.org/10.5772/intechopen.72872

This chapter describes, in Section 2, the global strategy used to design, implement and program the palletizer robot. The robot implementation and the A-start algorithm are explained in Section 3. Experimental results are presented in Section 4. Finally, Section 5 includes the conclusion and outlines future work.

2. Global strategy for palletizer robot navigation

The palletizer robot proposed here moves a box from place A to place B while navigating and avoiding collisions. To attain this goal, the main tasks involved and tackled here are as follows:

1. Perception of the environment by using sonar and contact sensors
2. Path planning strategy based on an A-star algorithm
3. Performing the robot trajectory and robot interacting with the dynamic obstacles

In general, the strategy programmed and performed by the palletizer robot is described in the block diagram of **Figure 1**. Both programming strategies DaNI and NXT are combined, but the control of the overall task is programmed on the mobile robot DaNI. The action "*compute trajectory to the box*" in the diagram uses the A-star algorithm and receives a predefined map of the environment with all static obstacles on it. This action is programmed on the DaNI robot, and it is performed in two stages; first, the robot moves to the box position and, second, the robot moves to the final pallet. Once the robot performs the first stage and arrives to the box position, the action "*set gripper ready*" involves the configuration and positioning of the gripper to take the box. This action was programmed on the NXT Lego. Once sensors indicate that the gripper has taken the box, the second stage of the trajectory is performed and the robot moves to the "*go to final pallet*" action. The robot locates the gripper and leaves the box carefully on the pallet. Then, the robot goes back to the initial position and the same process starts again if more boxes must be moved. Finally, dynamic obstacles are detected using the sonar sensor during robot mobile navigation and the robot stops if an obstacle is found in its path, and it continues its trajectory when the obstacle is not detected anymore.

2.1. Robot model description

The mobile robot used in this project is the robotic platform called NI LabVIEW robotics Starter Kit®, described in [8], also known as DaNI 2.0, developed by NI. This mobile robot was designed to develop and run algorithms in real time for autonomous system applications and can be programmed on two different languages: LabView or C.

Each wheel of the robot is connected to a DC motor which provides the traction force and stabilization wheel for balancing the robot; the kinematic model and the representation of robot position used in this work is the same as the one presented in [5]. The sbRIO-9632 card

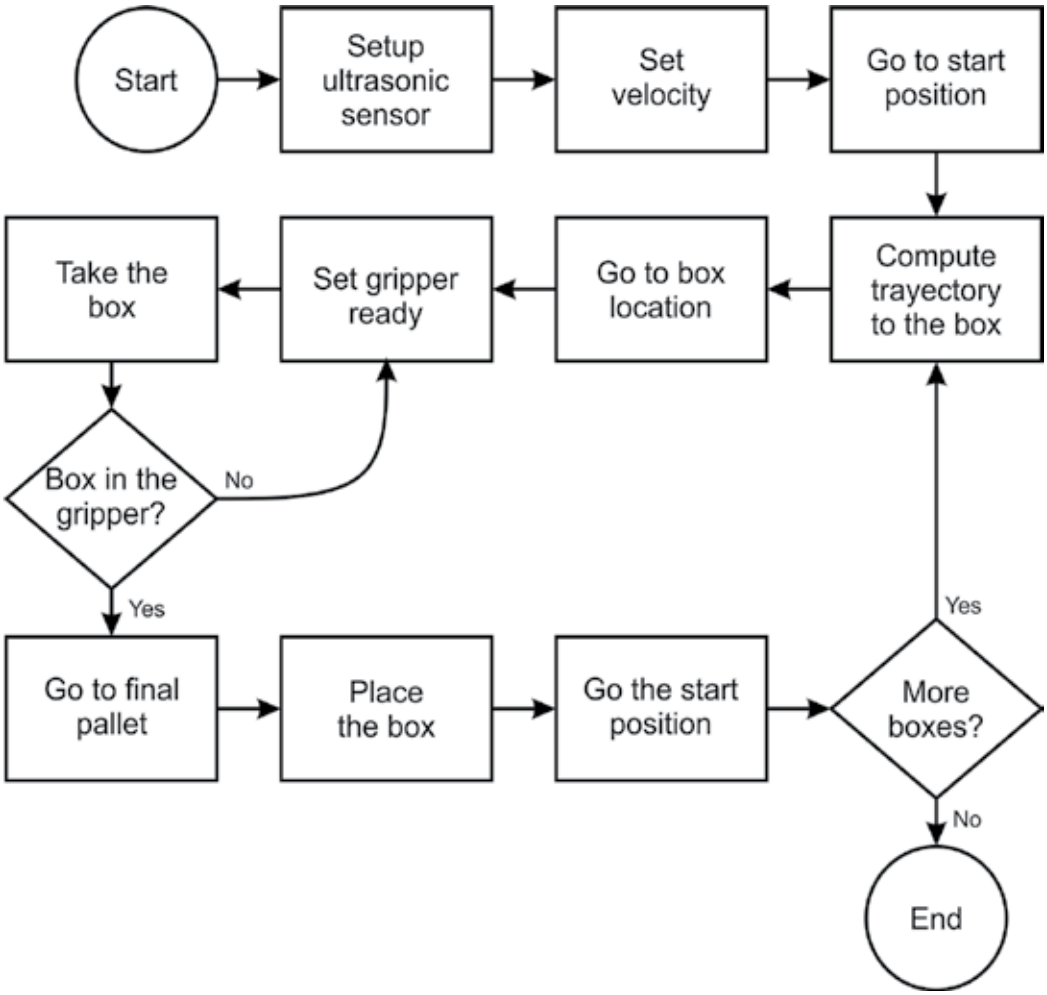

Figure 1. Global strategy for palletizer robot moves a box from place A to place B.

was developed by NI and contains a real-time processor which serves as a main control unit for the robot [10]. In addition, this platform includes a field-programmable gate array (FPGA) Xilinx Spartan-3 which is a reconfigurable device that executes programmed tasks in real time, that is, the active response of the system to external events. For this FPGA, a higher level of programming is possible using the NI LabVIEW® robotics software, which is a graphical language. Programming languages like C, C++ or Java could also be used.

This mobile robot is programmed using an efficient algorithm to cover a trajectory that takes it to the box that needs palletizing. The aim is to illustrate the function of a box palletizer robot in industry. Yet, at this stage, it is only a prototype to show basic functions not involving heavy weights as those handled by an industrial robot.

2.2. Robotic arm with LEGO MINDSTORM NXT 2.0

The Lego Mindstorm is a programmable robotic kit developed by Lego® and introduced for the first time in September 1998. The kit Lego Mindstorm NXT 2.0 provides basic pieces to construct mini-prototypes of robots by means of the assembly of mechanic plastic parts such as wheels, gears and bricks, among others, and electromechanic parts such as motors and different kind of sensors; finally, the robot prototype is programmed in an interactive way. The robots constructed using the Lego Mindstorm kit can simulate the same functionalities as real robots of this kind.

The robotic arm assembled for this project has a degree of freedom; its design is based on a parallel mechanism, and its motion is restricted only to vertical movements (up and down). The NXT brick is the central unit processing for programming robot task using LabVIEW Robotics which is the same graphical language used for programming the mobile robot DaNI. The communication between the NXT module and the PC is via Bluetooth.

2.3. Analysis of the gearbox

The mechanical design of the robotic arm is based on a parallelogram arm (four bars) [11] and an arrangement of gear wheels (called gear train) [12] to transmit turning force and to provide a degree of freedom. The four-bar mechanism consists of two vertical bars of 8 cm in height and two horizontal bars of 6 cm in length. To implement our parallelogram arm, the four-bar mechanism is implemented twice, one for each servomotor used to move the arm up and down. Each motor is finally fixed to the gear train.

A 3D model in Solidworks® [13] of the different ratios of used gears is shown in **Figure 2**. Note that different ratios are used for drive transmission and for the moving arm in accordance with the desired speed ratios, which will be explained below.

The number of teeth on the gear is the most important parameter during gearbox design, because the speed of a final gear train only depends on this parameter, with 24 and 36 teeth being the most common sizes used. With n and Z being the desired angular velocity and the number of teeth of a gear, respectively, both parameters are directly related as:

$$n_1 Z_1 = n_2 Z_2 \tag{1}$$

where index i represents the motor ($i = 1$) and the driven ($i = 2$) gear. This equation provides the number of teeth required to provide a given angular velocity. If the rate Z_1/Z_2 is less than 1, the speed will be reduced. In our case, the gearbox uses eight gears, and we consider $Z_1 = 8$ teeth for gears 1, 3 and 5, and $Z_2 = 24$ for gears 2, 4 and 6, yielding a ratio of *1/3*. The last two gears 7 and 8 are considered as $Z_1 = 16$ and $Z_2 = 36$, respectively, with a ratio of 4/9. **Figure 3** shows the 3D design of the gear train implemented.

Another important parameter during gear train design is the relation between power supply and torque of the servomotor. The Lego servomotor datasheet establishes that

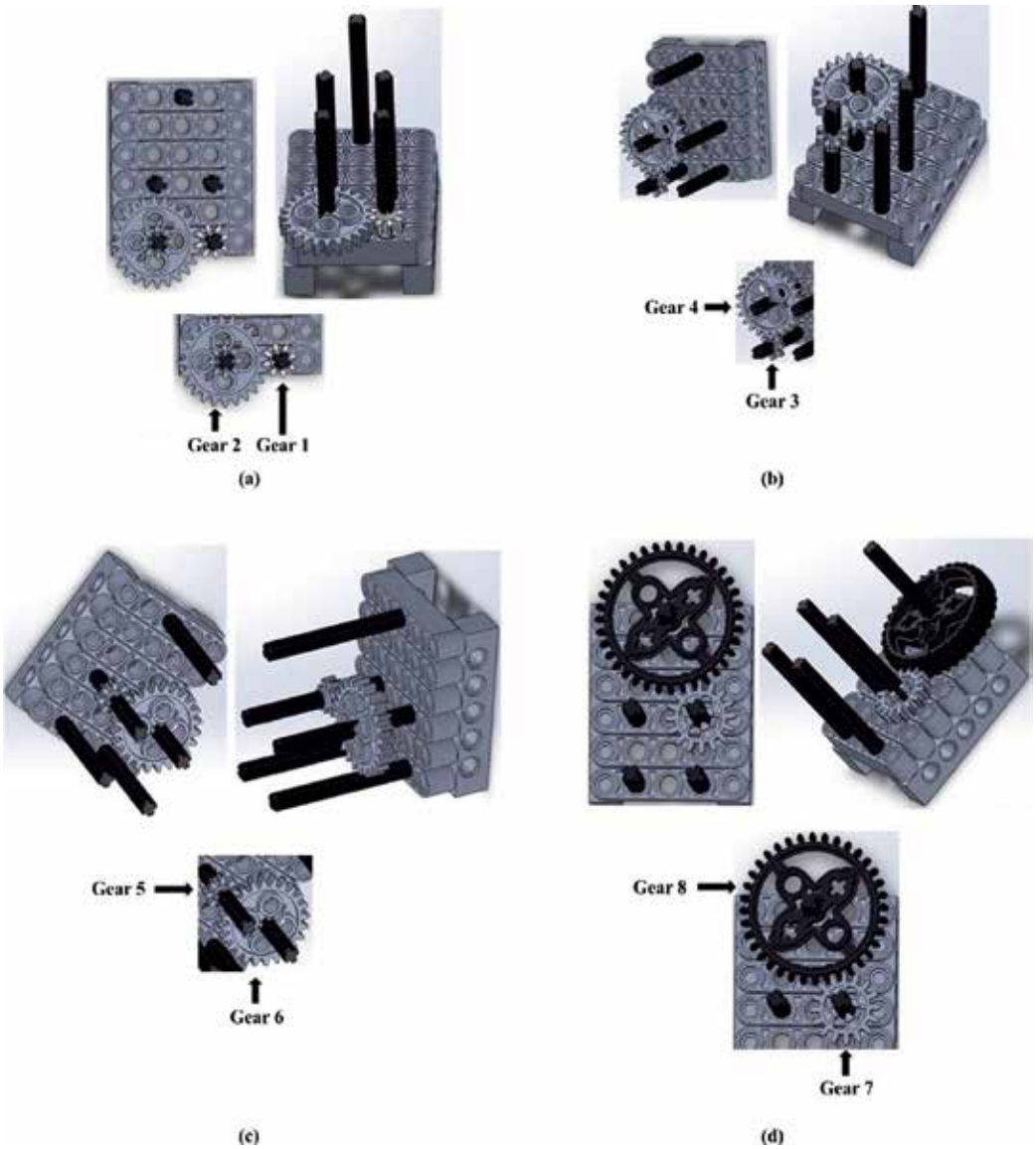

Figure 2. Ratios of the gearbox: three pairs of gears with a ratio of 1/3 are illustrated in (a), (b) and (c); in (d) the final pair of gears 7 and 8 has a ratio of 4/9.

for a power supply of 9 V (i.e., using 100%) the corresponding torque is 19 Ncm, and for 7.2 V (using only 75%) the torque is 16 Ncm [14]. For security reasons, we consider 15 Ncm as the maximal torque value provided by the robotic arm. In addition, **Table 1** shows experimental values of the angular velocity obtained at different values of power supply. Thus, considering a power supply of 75%, the angular velocity of the gearbox is around 95 rpm.

http://dx.doi.org/10.5772/intechopen.72872

On the other hand, with T_1 as the torque of the gearbox, then the power of the robotic arm is obtained by:

$$P = T_1 n_1 \tag{2}$$

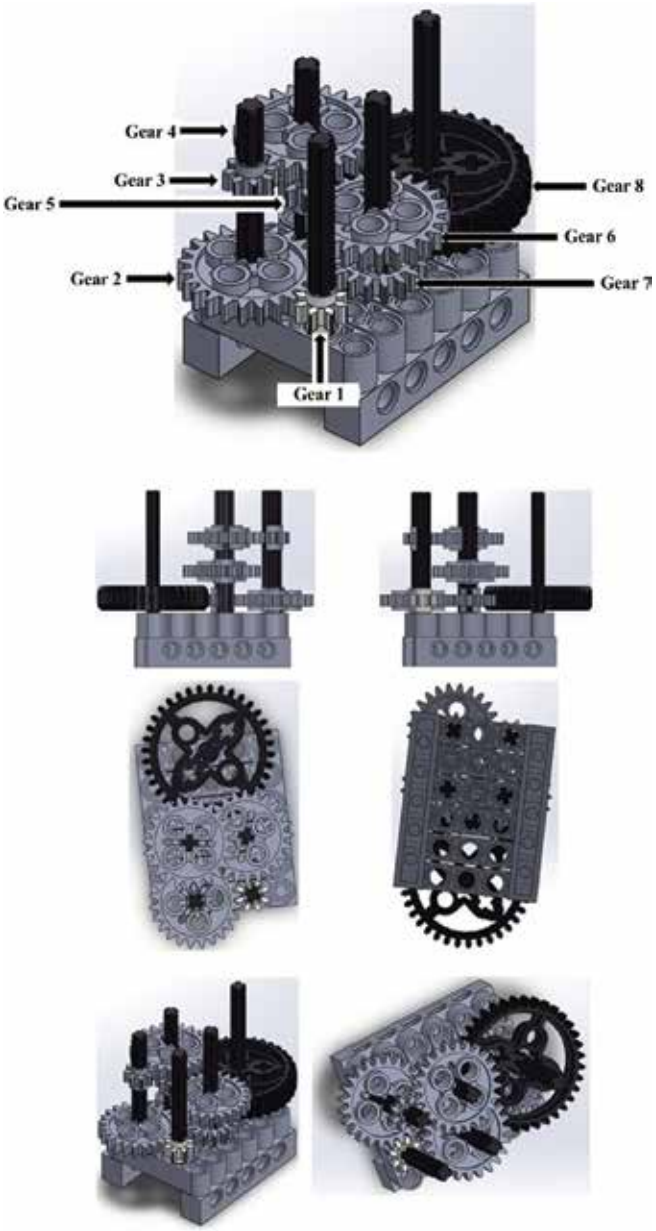

Figure 3. 3D design of the gearbox.

% Power supply	Angular velocity (rpm)
100	135.490
75	95.335
50	61.183
25	25.207

Table 1. Relation between power supply and torque of the servomotor (Lego datasheet [14]).

considering $T_1 = 0.15$ Nm and angular velocity of the motor as $n_1 = 9.9835$ rad/s; thus, the driven power of the gearbox is *1.4975* W.

To obtain the internal velocities along the gear train, we use the torque equation defined as:

$$T_2 = T_1 \cdot \frac{n_1}{n_2} \tag{3}$$

As the angular velocity n_1 is the same as the motor velocity, n_2 is given by:

$$n_2 = n_1 \cdot \frac{Z_1}{Z_2} = 95.335 \cdot (8/24) = 31.7783 \text{ rmp} \tag{4}$$

Then, the torque of gear 2 is obtained using Eq. (3), yielding

$$T_2 = T_1 \cdot \frac{n_1}{n_2} = 0.15 \cdot (95.335/31.7783) = 0.45 \text{ Nm} \tag{5}$$

Following this procedure, **Table 2** shows the torque values for 3–8 gears.

Therefore, torque and angular velocity at the output gear train are 9.1124 Nm and 1.5693 rpm, respectively. If the final torque is divided by the gravity force ($g = 9.81$ m/s^2), then we obtain the mass in kg that the gripper can carry if the robotic arm length were 1 m. In our case, the robotic arm length is 0.20 m, a value that represents one-fifth of the reference value 1 m. By considering that, torque increases in the same factor as the orthogonal length to the applied force decreases, and the final mass that our robotic arm of 0.20 m in length can carry is as follows:

$$0.20 \text{ m} \rightarrow 5(0.9289 \text{ kg}) = 4.6445 \text{ kg} \tag{6}$$

#Gear	Torque (Nm)	Angular velocity (rpm)
3	0.45	31.7783
4	1.35	10.5928
5	1.35	10.5928
6	4.050	3.5309
7	4.050	3.5309
8	9.1124	1.5693

Table 2. Torque and angular velocity of gearbox.

http://dx.doi.org/10.5772/intechopen.72872

3. Implementation of the palletizer robot

The robotic arm assembly requires bricks, girders, angle brackets, gearwheels, three servomotors and four touch sensors included in the Lego kit. Two of the servomotors move the mechanical part of the arm up and down, providing a degree of freedom. The third servomotor is used for closing and opening the griper. One of the touch sensors is at the base of the arm with the aim of sensing the lower position of the arm; similarly, a second touch sensor is located for sensing the higher position that can be reached by the arm. Third and fourth sensors are located on the gripper for measuring the opening and closing degrees controlled by the servomotor. A 3D model of the final robotic arm designed on Google SketchUP® [15] is illustrated on **Figure 4**, and the real robotic arm is shown in **Figure 5**; the gripper consists of four jaws to guarantee that object will be securely held.

The gearbox was designed in line with the required force and velocity to take and move an object from one place to another, without forcing the servomotors. The gearbox uses 8 gears: 3 of 16 teeth, 4 of 24 teeth and 1 of 36 teeth; it was designed following the analysis described in Section 2.3).

Once the robotic arm was built, we slightly modified the DaNI robot structure with the aim of mounting the robotic arm on it. In general, we replaced the rear omnidirectional wheel of

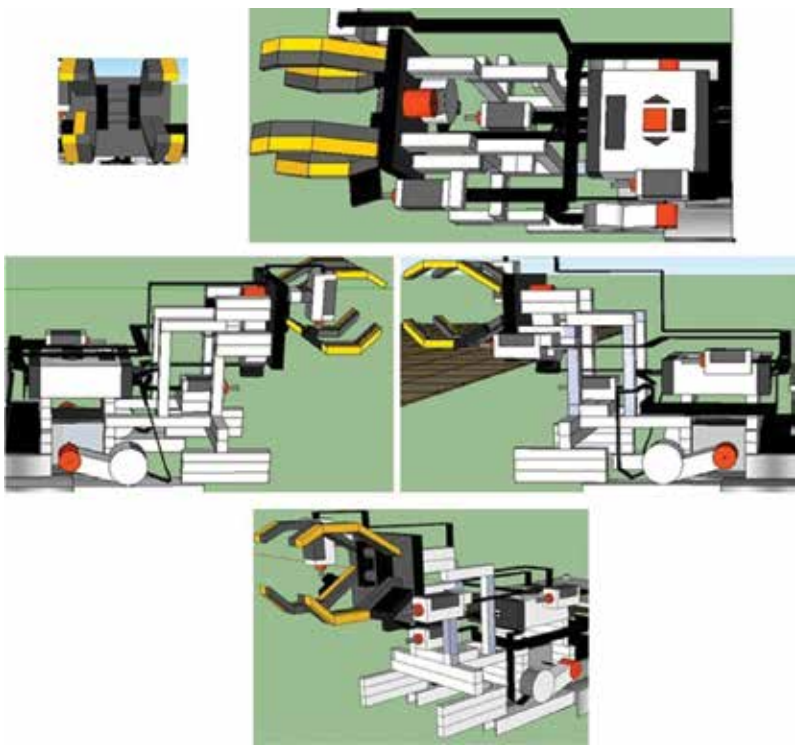

Figure 4. 3D model of the robotic arm designed on Google SketchUP®.

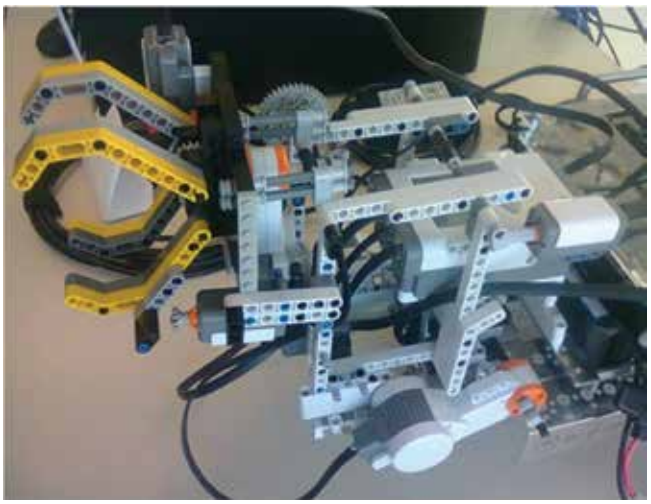

Figure 5. Robotic arm designed using Lego kit.

DaNI for a caster wheel of 1 inch. The wheel was fixed onto the chassis of DaNI creating free space to mount and fix the robotic arm. A 3D model of the mobile robot designed on Google SketchUP is shown in **Figure 6** and the real palletizer robot is shown in **Figure 7**.

3.1. A-star algorithm

The robot trajectory starts in an initial position from which the robot moves to the object location, then it goes to the pallet and, finally, it moves back to the initial position. The final path establishes horizontal and diagonal trajectories that represent the minimal costs to achieve the goal.

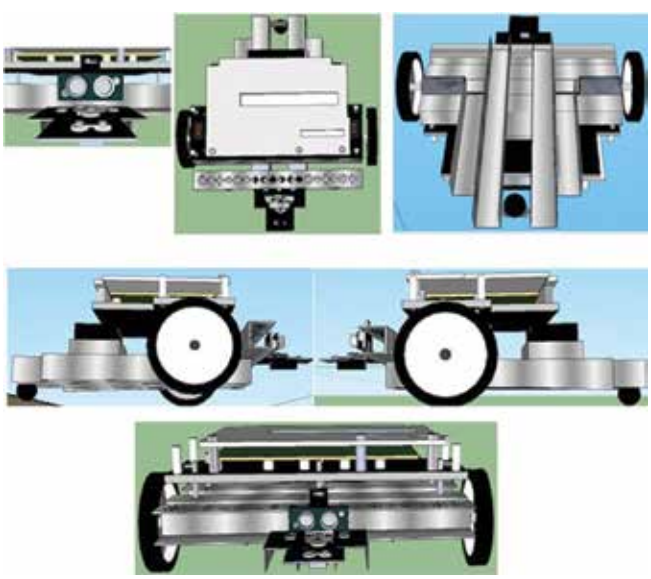

Figure 6. 3D model of the DaNI robot designed on Google SketchUP®.

http://dx.doi.org/10.5772/intechopen.72872

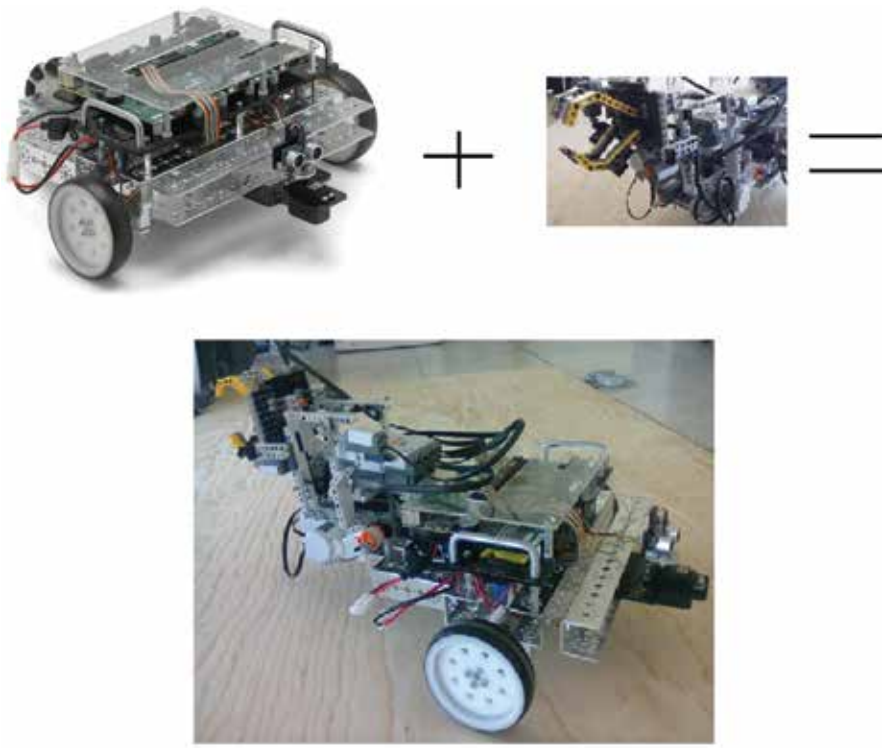

Figure 7. Final palletizer robot.

The planning technique used was the so-called A-star algorithm [16], which basically consists in the research of the best first trajectory that provides the shortest path from all possible roads. The final path is the union of partial movements that the robot must perform to get to the final position, that is, intermediate points before reaching the goal. Here, we use the term of nodes to refer these intermediate points that can be seen also as neighboring or adjacent points to the actual robot position. For each intermediate point, the A-star algorithm evaluates the next trajectories that could be reached based on minimal cost [17]. Thus, a final path guides the robot to a goal position warranting safe navigation.

Figure 8 illustrates a scene that we will use as an example of how to get from position A to B using the A-star algorithm. It is important to point out that two lists are needed to save the adjacent nodes: (1) open list for adjacent nodes that will be compared and (2) close list for nodes that cannot be considered anymore.

The first step to the A-star algorithm is to include the initial node position A to the close list. Thus, the iterative search starts including to the open list all reachable nodes from the initial node A, while unreachable nodes are, for instance, the occupied nodes. In accordance with **Figure 8**, eight nodes were included, depicted as squares and the arrows inside point to their parent node.

Eq. (7) computes the shortest path possible to the goal node B in the fewest moves.

$$F(x) = G(x) + H(x) \tag{7}$$

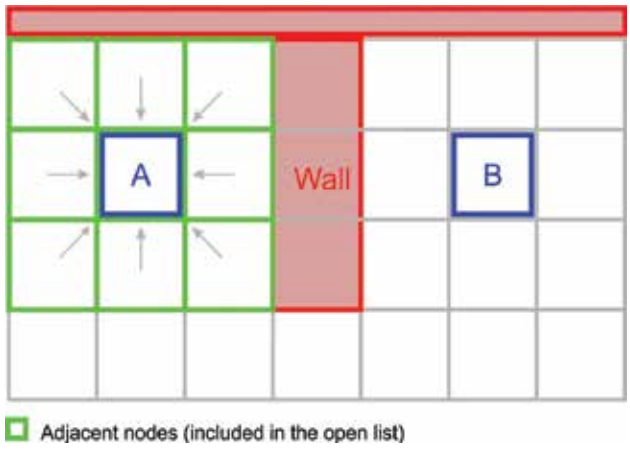

Figure 8. Environment to go from A to B.

where $F(x)$ = cost of the shortest path to the goal, $G(x)$ = cost of the movement from A node to an intermediate node and $H(x)$ = cost of the possible movement to go from an intermediate node to the goal.

Every node on the open list is evaluated using this equation, and the node with the lowest $F(x)$ value is chosen. The value $G(x)$ represents the costs involved in reaching the neighbor nodes. Horizontal or vertical nodes are weighted as 10 (because there is one node to get there). Then, diagonal moves are weighted as 14 because this is the closest without choosing a diagonal. $H(x)$ value is estimated using the Manhattan distance, but many other distances can be used as well. Such distance considers only vertical or horizontal moves to get to the final goal from any intermediate node, ignoring diagonal moves and obstacles on the way. This does not imply that all the environment must be free of moving or static obstacles; it is just the real physical distance between two nodes, after a second weighted value excludes possible paths with occupied intermediate nodes. The punctuation after computing Eq. (7) in our example is shown in **Figure 9(a)**. As the method does not consider obstacles, the shortest distance from B to right node of A is $H = 30$ because this node is reachable in three nodes. Similarly, the upper right

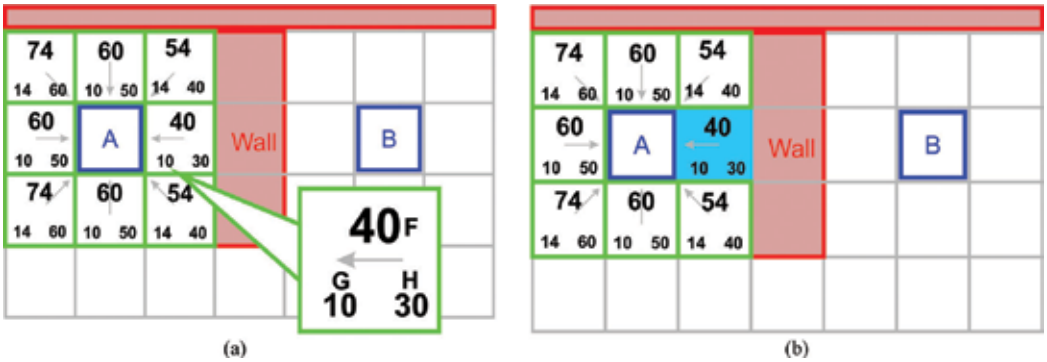

Figure 9. A-star algorithm. (a) First iteration of the algorithm and (b) the lowest cost node.

corner node of A is $H = 40$ because four nodes are required to get there from B. Thus, the lowest value of F is 40 being the right node chosen as the first move that must be performed by the robot (see **Figure 9(b)**). Also, this node is added to the close list to avoid considering it again.

During the next iterations of the algorithm, the new initial position is the node resulting from the last iteration; so all previous nodes in the open list are moved to close list, and the open list will contain the neighbor nodes to such initial node. In addition, the values of $F(x)$, $G(x)$ and $H(x)$ are updated to search for the next low cost node. In our example, only four nodes are added to the open list corresponding to up, down and diagonal nodes to the node under evaluation. Thus, the next node added is the lower right corner node, as it throws the lowest value of $F(x) = 54$.

Once again, the iterative process starts by adding new neighbor nodes to the open list and computes the cost function to find the closest node in the open list with the lowest cost. This recursive process ends when the final node is added to the close list. **Figure 10(a)** shows the nodes added to the close list after five iterations of the A-star algorithm in blue. On the same figure, image b shows the final shortest path possible, starting from the goal B and going to the parent node backwards.

3.2. Programming the palletizer robot

As mentioned before, the NXT brick can be programmed in LabVIEW code to perform the robotic arm movements:

1. Move up and down the arm.
2. Open and close the gripper to take the box.
3. Open and close the gripper to leave the box on the pallet.

Moreover, to perform a safe and coherent navigation, the path planning algorithm is programmed on the sb-RIO reprogrammable card of the robot, that is, following the abovementioned A-star technique. In this chapter, we explain robotic arm motion and pathfinder algorithm.

The LabVIEW Robotics software provides a module of the A-star algorithm which computes an optimal path to get to goal position [18]. The path includes horizontal and vertical trajectories of the robot, left or right rotations and diagonal displacements. However, our mobile robot can only

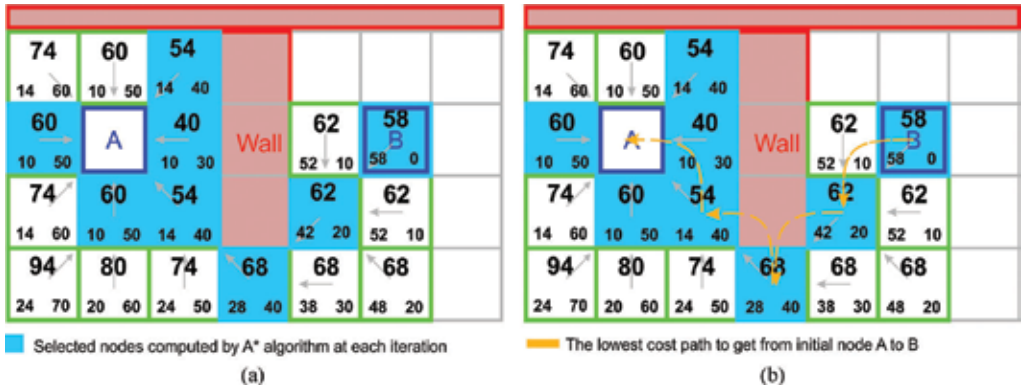

Figure 10. (a) Nodes computed by the A-star algorithm at each iteration and (b) path to get from node A to node B.

perform horizontal and vertical trajectories and left or right rotations. Diagonal movements cannot be performed, because the robot size is not considered for the final path computation, and, consequently, it is more important to avoid possible robot collisions than to get an optimal traversing path. For this reason, the virtual instrument (VI module) provided on the software is modified to exclude all diagonal nodes that could be included in the path. The modified class is called "*GetNeighbours*" of the library "*OccupancyGridWorldMap,"* and only we establish a fixed threshold value for the cost function of the diagonal nodes, such that it will always be higher than other possible nodes. **Figure 11(a)** illustrates simulation results of the A-star algorithm, and **Figure 11(b)** the modified version for the same environment before programming the mobile robot DaNI. Note that there are diagonal displacements on the path, yet they are not too close to the obstacles, so the risk of collision is minimal. Real tests of the final palletizer robot are presented in the next section.

4. Experimental results

A 3D model of the palletizer robot designed on Google SketchUp® is shown in **Figure 12**.

The experimental tests were performed on room with natural light, without any kind of obstacles around. The robot moves on a rectangular wood base sized 2 × 1.2 m, and the pallet is a square wood base of 30 cm located on the upper right corner of the base. Initially, the robot is located at the lower left corner as its start point. This information about the environment is registered as a matrix in a text file, with values '0' and '1' representing free and occupied cells in the environment, respectively. The size of the matrix is related to the navigable space, and in our case, the matrix is 20 × 12; therefore, each cell is 10 cm big, representing an environment of 2 × 1.2 m.

Some images of the robot arriving at the object location and picking it up with the gripper are included in **Figure 13**. Many orders are carried out during this first routine of the robotic arm: set, down, open, close and up gripper to take the object. The second routine of the robotic arm consists in placing the object on the final pallet. This routine starts with the arm up, and then, it goes down slowly. Once the gripper is located over the pallet, it opens, goes up and finally closes the gripper (**Figure 14**).

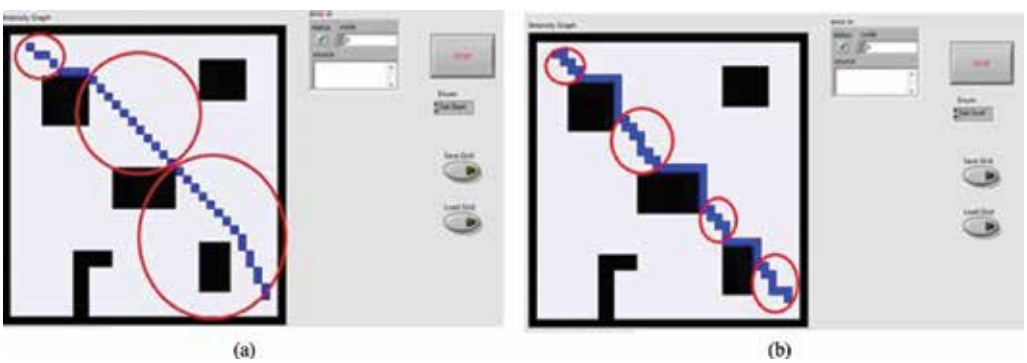

Figure 11. (a) Simulation results of the A-star algorithm and (b) results of modified A-star algorithm.

http://dx.doi.org/10.5772/intechopen.72872

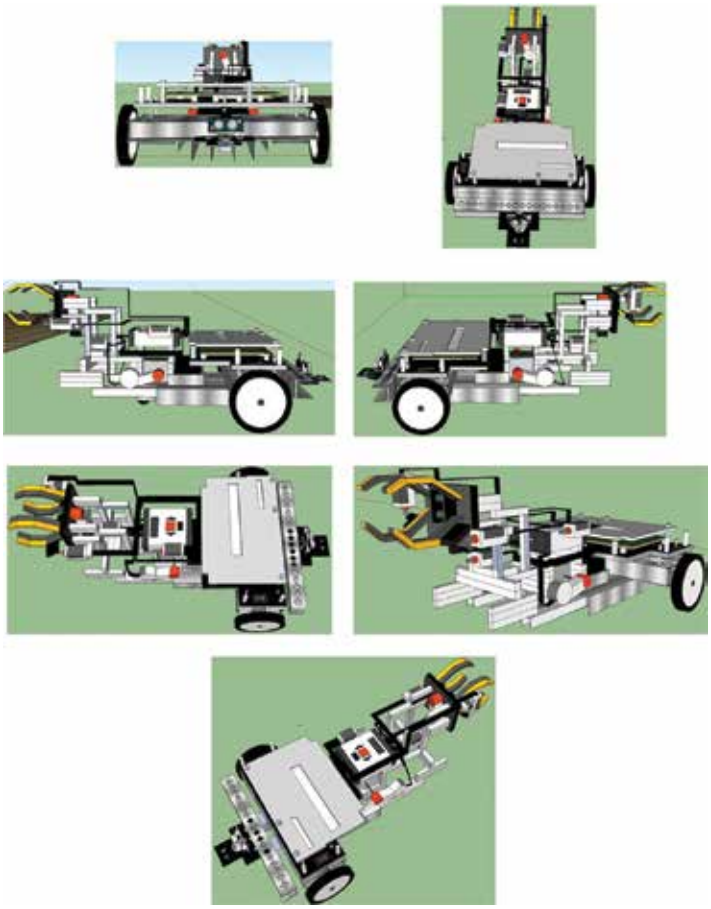

Figure 12. 3D model of the assembled palletizer robot.

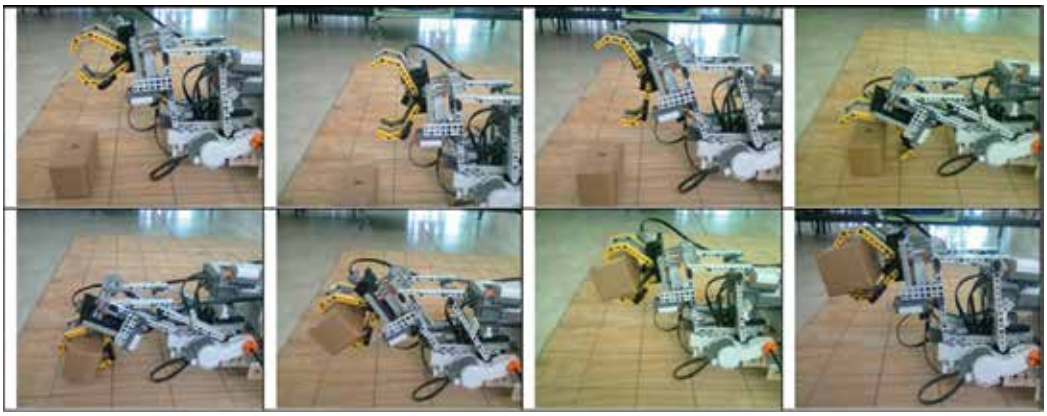

Figure 13. Images illustrate the first routine performed by the robotic arm: picking up an object.

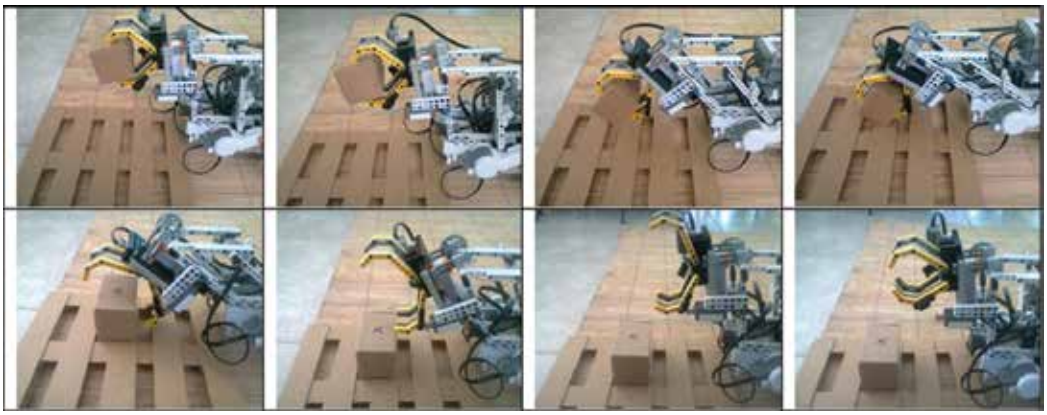

Figure 14. Images illustrate the second routine performed by the robotic arm: placing object on the pallet.

The robotic arm motion is a program synchronized with the path planning strategy in the mobile robot; in contrast, the A-star algorithm was established as the main routine of the palletizer robot. That is, rather DaNI robot or NXT Lego performs their programmed tasks sequentially and the mobile robot has the master control. The path planning module on the robot consists of three stages: initial, intermediate and final. The first stage involves the A-star algorithm: it receives the initial map of the environment (the text file explained above), the start and end positions of the robot, and then, the module calculates the navigable path of the robot. In addition, this stage calculates an occupation map, indicating the cells that the robot must "visit" during navigation to get to the goal position.

In the intermediate stage, the occupation map with the resulted trajectory is analyzed and adequate in accordance with the robot dimensions and motor power: set velocity parameters, rotation angles, and warranty that all the movements could be performed by the DaNI robot. The interpretation and adequacy of the path in the robot consists in setting up when the robot moves straight, turns left or right or when the goal is reached. Thus, a straight motion is included (represented as '0') when the coordinates (x,y) of any pair of consecutive cells in the path change only in the 'x' or 'y' value. If both coordinates change, then a left or right motion ('1') is included depending on the change in the coordinates. As the cells in the path are evaluated in pairs of cells, the robot stops when the second (x,y) cell coordinates is the goal, represented as '2'. In this way, at the end of the intermediate stage, a chain of instructions that includes numbers '0', '1', '2' codes the physical movements that the robot must perform.

During the final stage, the movements and rotations planned for the robot are carried out in accordance with the chain of instructions provided by the intermediate stage. **Figure 15** illustrates a graphical interface that simulates robot navigation in an environment of 36×36 cells with obstacles, for an initial point and final point of $(1,1)$ and $(20,20)$, respectively. The output path is shown in the left graph, while the output data of initial, intermediate and final stages, including the chain of instructions, are illustrated at the top of the graphical interface.

http://dx.doi.org/10.5772/intechopen.72872

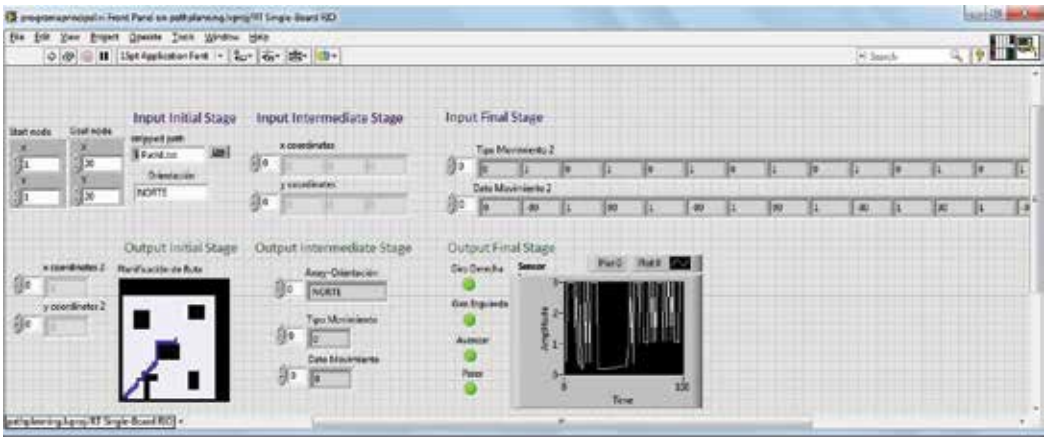

Figure 15. Graphical interface of the robot navigation.

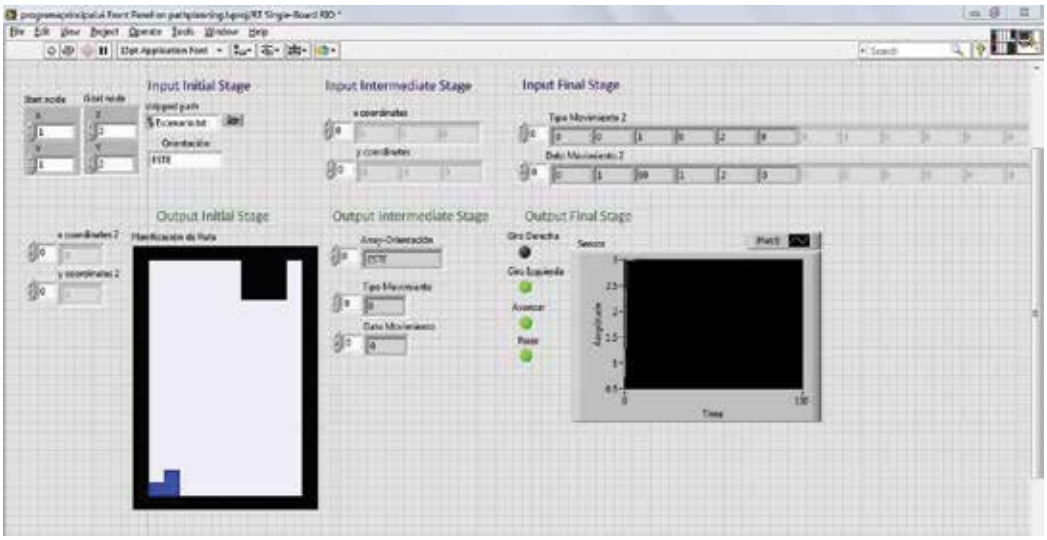

Figure 16. First path to move the robot from (1,1) initial position to (2,2) box location, without obstacles in the environment.

In our real environment, the first test carried out was to move the robot from (1,1) cell (initial robot position) to (2,2) cell (box location), without obstacles on the map (**Figure 16**). **Figure 17** shows the first part of the proposed routine, when the robot takes the box.

The second path performed consists of getting to a second final point, that is, the pallet location for leaving the box. Here the initial position will be the object location which corresponds to the final coordinate performed in the first stage. Thus, considering the pallet location as cell *(8, 3)*, the second path performed to get the pallet is illustrated in **Figures 18** and **19**: graphical interface results and the second part of the real robot performance.

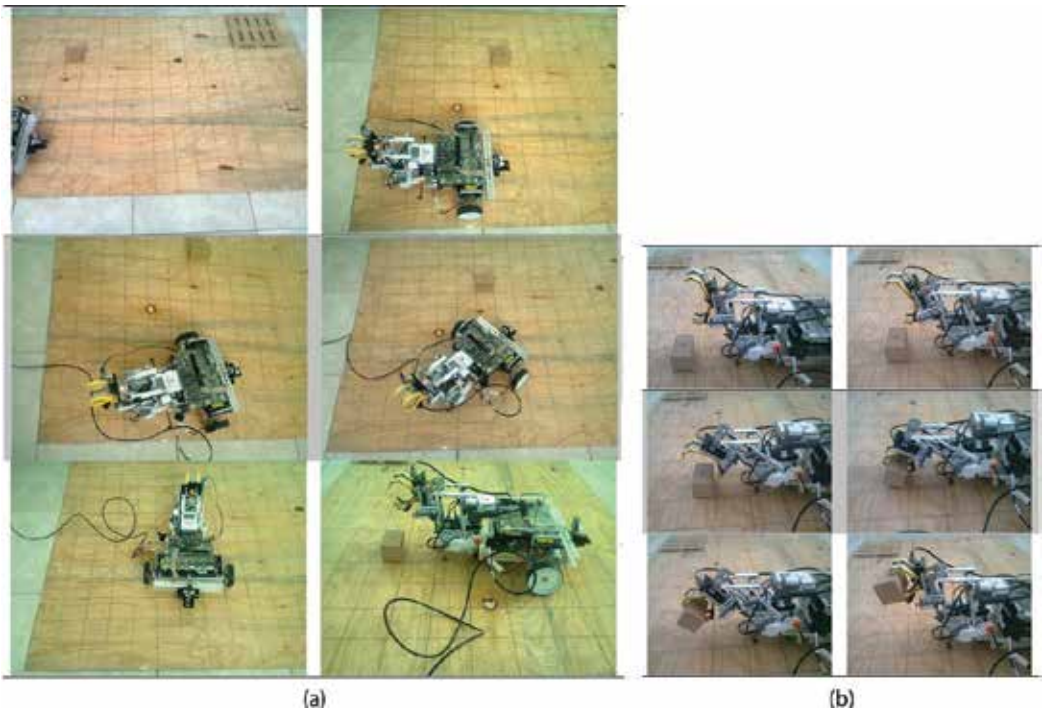

Figure 17. Real robot performance. (a) Robot path to the box location and (b) robot takes the box.

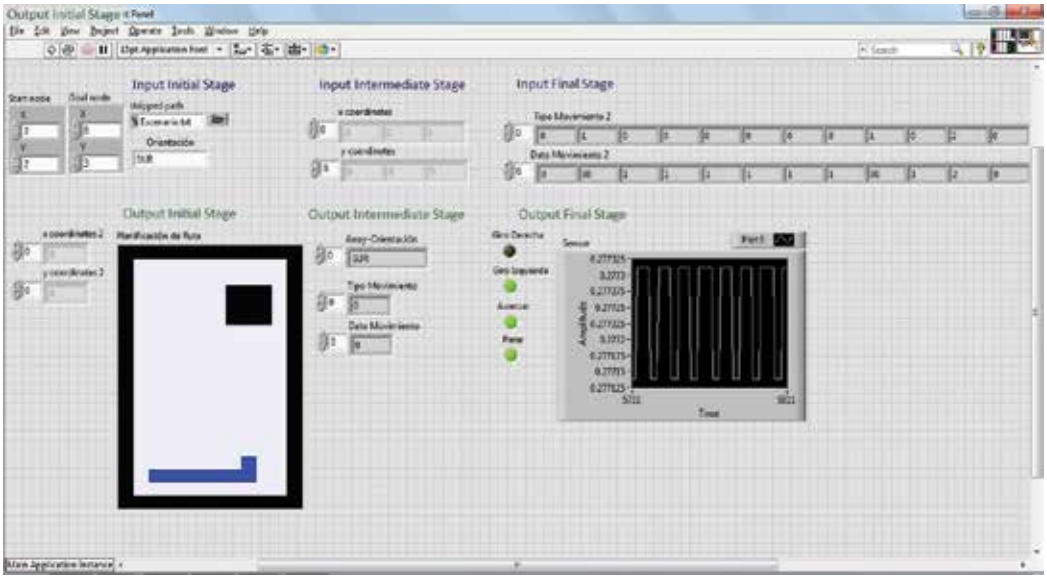

Figure 18. Path to move the robot from (2,2) to (8,3) cells, without obstacles in the environment.

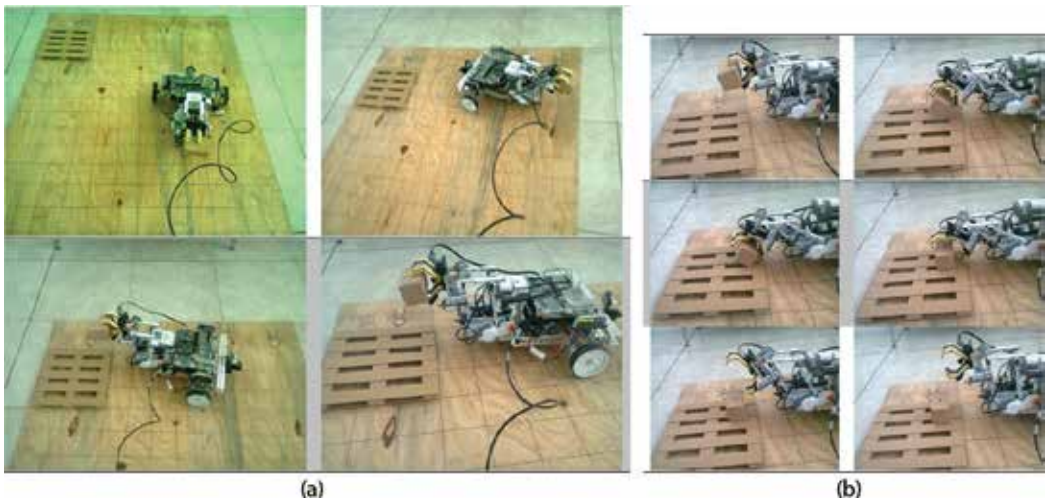

Figure 19. Final part of the routine. (a) By rows: robot moves from initial position to the pallet carry on the box and (b) by rows: robot leaves carefully the box on the pallet.

If no more boxes must be palletized, the robot performs a third path to reach the initial position again, in our example cell (1,1).

5. Conclusions and perspectives

The task of moving an object from one place to another in an autonomous way requires many considerations: the mobile robot, the mechanism used to carry the object, the path planning strategy, and synchronization of all the systems involved in the task. In this chapter, the robotic NXT arm can be programmed at a maximal distance of 10 m. and some of the considerations to implement this kind of project are:

1. Validate ultrasonic sensor values when executing the trajectory.

2. Execute the trajectory in two stages: to reach the box and to place it in the pallet.

An appropriate space must be found for robot navigation purposes to avoid excessive or deficient friction of the wheels against the floor, to establish speeds and distances.

The overall strategy programmed on the mobile robot should encourage beginners or young people interested in robotic developments. We are currently working on the integration of a method to verify if the second phase of the trajectory can be performed as initially established. In addition, we are also correcting odometric mistakes at a mechanical level. On both platforms, DaNI robot and NXT, LabView and LabView robotic modules are used respectively to

program communication between the robotic arm and the mobile robot using the same rules of code. Finally, a perspective to increase the accuracy of the navigation consists in adding a video camera to monitor the palletizing process.

Author details

Dora-Luz Almanza-Ojeda*, Perla-Lizeth Garza-Barron, Carlos Rubin Montoro-Sanjose and Mario-Alberto Ibarra-Manzano

*Address all correspondence to: luzdora@ieee.org

Electronics Engineering Department, DICIS, University of Guanajuato, Salamanca, Guanajuato, Mexico

References

[1] Spong MW, Vidyasagar M. Robot Dynamics and Control. India: Wiley India Pvt. Limited; 2008. https://books.google.com.mx/books?id=PtxYAv7ZUYMC ISBN: 9788126517800

[2] Iocchi L, Ruiz-Del-Solar J, Zant T. Advances in domestic service robots in the real world. Journal of Intelligent and Robotics Systems. 2014;**76**(1):3-4. DOI: 10.1007/s10846-014-0021-1

[3] Zavadskas EK. Automation and Robotics in Construction: International Research and Achievements. Universidad de Alicante; 2012. pp. 2-5. DOI: 10.1016/j.autcon.2009.12.011

[4] Lopez BL. Distribuciones hibridas: Los sistemas de fabricacion flexible [Internet]. Available from: http://www.academia.edu/10375825/DISTRIBUCIONES_HIBRIDAS [Accessed: 02-08-2017]

[5] Almanza-Ojeda DL, Gomar-Vera Y, Ibarra-Manzano MA. Occupancy Map Construction for Indoor Robot Navigation. In: Hurtado EG, editor. Robot Control. Croatia: Intech; ch. 4. pp. 69-87. ISBN: 978-953-51-2684-3, September 2016. http://www.intechopen.com

[6] "VexRobotics" [Internet]. Available from: http://www.vexrobotics.com.mx/ [Accessed: 09-08-2017]

[7] Zowi robot in Europe [Internet]. Available from: http://zowi.bq.com/fr/ [Accessed: 09-08-2017]

[8] National Instruments Corporation. Introduction to the LabVIEW Plataform [Internet]. Available from: http://www.ni.com/webcast/439/es/ [Accessed: 29-09-2017]

[9] Hopkins B, Antony R. Bluetooth for JAVA. USA: Apress; 2003. ISBN: 978-1-59059-078-2

[10] National Instruments Corporation. NI Single-Board RIO Embedded Control and Acquisition [Internet]. October 2012. pp. 1-3. Available from: ftp://ftp.ni.com/pub/branches/northern_region/fpga_kit_feb14/what_is_ni_singleboardrio.pdf [Accessed: 29-09-2017]

[11] Rico JM. Analisis dinamico de un mecanismo plano de cuatro barras. Analisys of Mechanisms, undergraduate course. Division de Ingenierias Campus Irapuato-Salamanca, Universidad de Guanajuato; Mexico. 2014. pp. 2-6

[12] Lent D. Analysis and Design of Mechanisms. USA: Prentice Hall; 1970. ISBN: 978-0-13032-797-0

[13] SolidWorks Corporation. Student's Guide to Learning SolidWorks® Software. France: D'assault Systemes SolidWorks Corporation, PMS0119-ENG; 2011. Available from: https://www.solidworks.com/sw/docs/Student_WB_2011_ENG.pdf [Accessed: 23-06-2017]

[14] Lego Mindstorm NXT Datasheet. Lego Mindstorm Education; Denmark. 2008. 66 p. Available from: https://www.generationrobots.com/media/Lego-Mindstorms-NXT-Education-Kit.pdf

[15] Trimble: How to use Google-sketchup, Trimble Corporate and SketchUp developers. Available from: https://www.sketchup.com/learn/videos/826 [Accessed: 14-08-2017]

[16] Dechter R, Judea P. Generalized best-first search strategies and the optimality of A*. Journal of the ACM. 1985;**32**(3):505-536. DOI: 10.1145/3828.3830

[17] Ferguson D, Likhachev M, Stentz A. A guide to heuristic-based path planning. Proceedings of ICAPS Workshop on Planning under Uncertainty for Autonomous Systems; 2005. www.cs.cmu.edu/afs/cs.cmu.edu/Web/People/maxim/files/hsplanguide_icaps05ws.pdf

[18] Gretlein S. Presentando labview robotics: de la fantasia a la realidad. Instrumentation Newsletter. 2010;**22**(2):3-5. Available from: ftp://ftp.ni.com/pub/gdc/tut/abril-junio_2010.pdf

Edited by Rastislav Róka

The book *Advanced Path Planning for Mobile Entities* provides a platform for practicing researchers, academics, PhD students, and other scientists to design, analyze, evaluate, process, and implement diversiform issues of path planning, including algorithms for multipath and mobile planning and path planning for mobile robots. The nine chapters of the book demonstrate capabilities of advanced path planning for mobile entities to solve scientific and engineering problems with varied degree of complexity.

Published in London, UK

IntechOpen

ISBN 978-1-83881-400-7