Universitat de les Illes Balears
Departament de ciències Matemàtiques i Informàtica

New Reactive and Path-Planning Methods for Mobile Robot Navigation

Javier Antich Tobaruela

Thesis supervisor

Dr. Alberto Ortiz Rodríguez

Dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor en Informàtica

Systems, Robotics and Vision group

Palma de Mallorca, February 2012

New Reactive and Path-Planning Methods for Mobile Robot Navigation

Javier Antich Tobaruela

Student's signature

Palma de Mallorca, February 2012

This is an administrative document written in Catalan

D. Alberto Ortiz Rodríguez, Doctor en Informàtica per la Universitat de les Illes Balears i Titular d'Universitat de l'àrea de Arquitectura i Tecnologia dels Computadors del Departament de Ciències Matemàtiques i Informàtica de la Universitat de les Illes Balears

FA CONSTAR:

que la present memòria *New Reactive and Path-Planning Methods for Mobile Robot Navigation* presentada per Javier Antich Tobaruela per optar al grau de Doctor en Informàtica, ha estat realitzada sota la seva direcció i reuneix la suficient matèria original per ser considerada com a tesi doctoral.

Signatura del director

Palma de Mallorca, Febrer 2012

Acknowledgements

Let me write the following words in Spanish. Muchas han sido las personas que a lo largo del desarrollo de esta tesis me han apoyado de una u otra forma. De una manera especial, quiero expresar mi agradecimiento a:

Ana	por estar siempre a mi lado, y enseñarme que no todo es el trabajo.
mi hijo **Pablo**	por robarme el corazón.
mi Madre	por su infinita bondad hacia los demás.
mi Padre	por no dudar de mi.
mi hermana María José, mi sobrino Marc, David, Miriam, y Sofía	por alegrarme cada minuto que he pasado con ellos.
mi hermana Cristina, Miquel, y la pequeña Daniela	por ser parte de mi vida.
mi cuñada Victoria	por su apoyo a Ana y el cariño mostrado hacia Pablo.
Alberto	por tantas y tantas cosas. Entre ellas: por enseñarme a investigar de forma rigurosa, por mantener su confianza en mi, y ser comprehensivo en los difíciles momentos que he vivido estos últimos años.
Julián	por meterse continuamente conmigo, y darlo absolutamente todo como pareja de bádminton.
Toni, Guillermo, José, Francesc, y Manuel	por todo lo compartido como docentes y doctorandos.
Biel, y Yolanda	por saber que siempre se puede contar con ellos.

Gabi, Fernando, Manolo, Iván, José Luis, y Pedro	por ser mis amigos desde hace más de veinte años.
Ángel	por su alegría contagiosa.
Jaume, Miquelet, Dani, Xavi, Llorenç, Xema, y Toni Sola	por hacerme sentir como si nunca me hubiera ido del antiguo centro de cálculo.
Oscar	por permitirme discutir con él conceptos y formalismos matemáticos, además de todo aquello relacionado con el complicado trabajo de ser padre.
Xisco y Emilio	por ser buena gente, y guardarme mis tortitas de maíz.
Antonio Teruel	por ser como es.
Pere, Marc, Andrés, Emili, Narcís, y David	por acogerme fantásticamente bien en la Universitat de Girona, y permitirme realizar mis primeros experimentos con vehículos submarinos reales.
Javier Mínguez	por ser cercano, y estar dispuesto a escuchar y debatir ideas.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The first part of this chapter introduces some basic concepts of the robotics field. To be more precise, we first define the term mobile robot (section 1.1), afterwards, we explore the different types of mobile robots (section 1.2), and, at last, we describe the paradigms of control that are most commonly used to achieve autonomous navigation in a mobile robot (section 1.3).

In the second part of this chapter, we outline the objectives and the structure of this dissertation (section 1.4).

1.1 Basic Terminology

The term *robot* comes from the Czech word *robota*, meaning drudgery or slave-like labour. It was first used to refer to the artificial workers made in a factory in a science fiction play produced by Karel Capek in 1921 called Rossum's Universal Robots (R.U.R.). This general idea has, however, evolved over the years in line with the advances in the robotics field. Nowadays, according to [1], a robot is defined as:

> "a machine able to extract information from its environment and use knowledge about its world to act safely in a meaningful and purposive manner";

or in other words, a generally autonomous physical system which can both sense its environment and act on it with the ultimate aim of achieving some user-defined goals.

By extension, *robotics* is the discipline that involves:

- the design, manufacture, and control of robots through programming to solve problems,

  and

- the psychological and biological study of the behavior of human beings and animals as well as the application of the resulting models to the design, manufacture and control of robots.

1.2 A Classification of Mobile Robots

Mobile robots are broadly classified into three types: ground, underwater, and flying vehicles (figure 1.1 gives an example of each of them). In the next lines, only the general aspects of the two first types of robots will be briefly discussed (the reason for excluding flying robots from the discussion is that this dissertation only reports experiments with ground and underwater robots).

(a) (b) (c)

Figure 1.1: An example of ground (a), underwater (b), and flying (c) robot.

(a) (b) (c)

Figure 1.2: An example of wheeled (a), tracked (b), and legged (c) robot.

1.2.1 Ground Vehicles

The simplest case of ground vehicle is a wheeled robot, as illustrated in figure 1.2(a). These robots comprise at least a driven wheel having optional passive / caster wheels and maybe even steered wheels. Most designs require two motors for driving and steering the mobile robot. Figure 1.2(a) shows a good example in that respect called *differential drive*, where two driven wheels allow the robot to go straight, to follow a curve, or to turn on the spot.

One important disadvantage of all wheeled robots is that they need some sort of flat surface for moving. In this sense, tracked robots are more flexible because of their capability to successfully navigate over rough / rocky terrains (look at figure 1.2(b)). To gain stability, these robots exert high-friction turns, as a consequence of the multiple points of contact of the tracks with the surface. As a last comment, notice that the large majority of tracked robots move along two parallel tracks, each driven by a separate motor.

Just like tracked vehicles, legged robots are also able to navigate through rough surfaces: for instance, climbing up and down stairs. Many different designs have been proposed for this kind of robots whose main difference is the number of legs. As a generally-accepted fact, the problems of making these robots balance and walk simplify as more legs are available, although at the expense of a higher cost, weight, and power consumption. Figure 1.2(c), by way of example, depicts one of the most advanced humanoid robots called ASIMO by Honda.

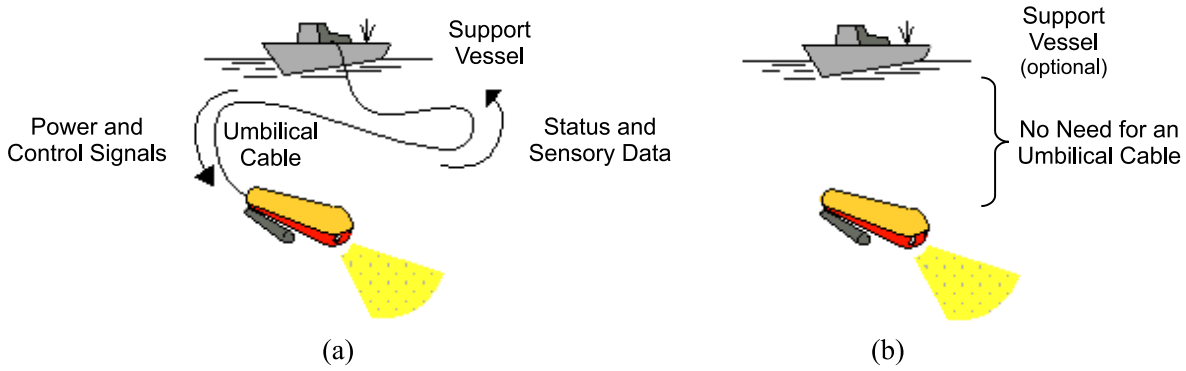


Figure 1.3: Operational setting for an ROV (a) and an AUV (b).

1.2.2 Underwater Vehicles

Several types of underwater vehicles are in use today. Among them, the two following deserve to be highlighted: the *Remotely Operated Vehicle (ROV)* and the *Autonomous Underwater Vehicle (AUV)*[1]. Their main features are summarized next (see also figure 1.3):

- ◇ An *ROV* is an unmanned underwater robot whose control is the responsibility of an operator who typically remains on a support vessel. An umbilical cable links the vehicle to a remote control console managed by the operator. Both electric power and control commands are sent down this cable, whereas data from the vehicle's sensors, such as video cameras and sonars, are sent up. In most cases, the robot is fitted with one or two arm manipulators in order to let it act on the environment.

  The first *ROV* was built in the late 1950s. However, commercial use of this technology did not start until the mid 70s; shortly after, its use was commonplace. Several thousands of vehicles have been built since then and are currently put into use by scientific, military and commercial organizations. Recently, an *ROV* owned by the Japanese Marine Science Technology Center has reached the bottom of the Challenger Deep in the Mariana Trench, the deepest part of the ocean (approximately 11033 metres).

- ◇ As compared to an *ROV*, an *AUV* is also an unmanned but self-sufficient underwater robot. In essence, this means that the vehicle carries its own energy source —batteries— and is programmed with a set of instructions that enable it to perform a mission without the assistance from an operator on the surface. These instructions include, among others, procedures for navigating between predetermined geographic positions while safely avoiding obstacles, as well as actions to be taken in case of equipment breakdown.

  In the past 30 years, nearly two hundred *AUVs* have been built, being, most of them, experimental. Despite this fact, they have achieved impressive results which is currently creating a demand for their use.

[1] The term *Untethered Underwater Vehicle (UUV)* is also used to refer to this kind of vehicles

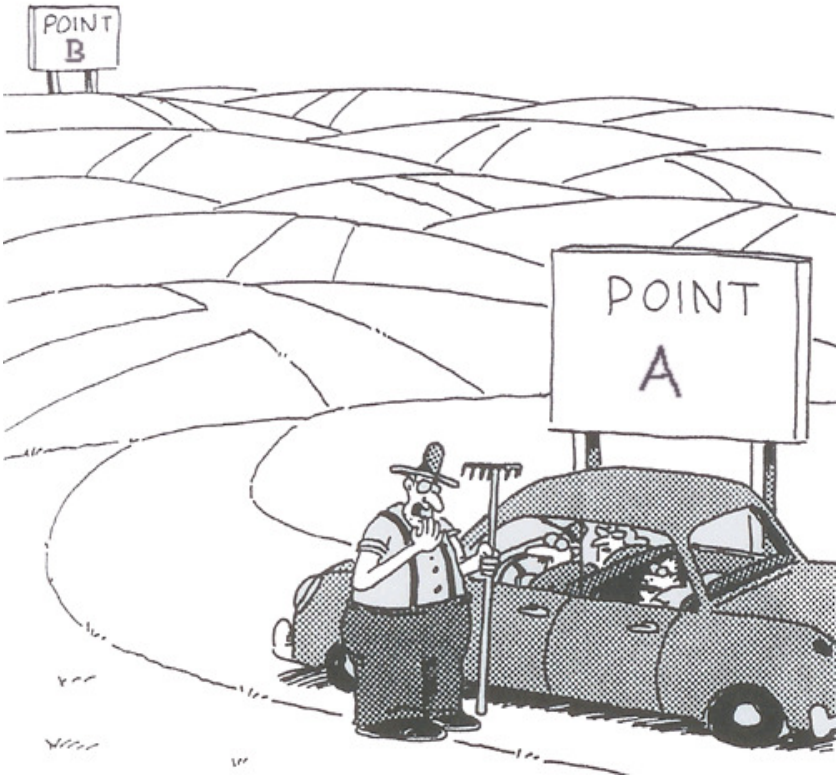


Figure 1.4: The problem of *navigation* consists of safely moving from point A —or the starting point— to point B —or the destination point.

1.3 Paradigms in Robot Control and their Application to Navigation

Let us start with two definitions:

- ◇ *Robot control* is the process of acquiring information about the environment —through the vehicle's sensors, or by using an a-priori model of the environment, or by combining both of them—, processing it as necessary to decide how to act, and then executing those actions by means of the available effectors to achieve, in an autonomous way, the set of goals corresponding to a user-specified mission.
- ◇ *Navigation* is the process of moving the robot from one place to another without colliding with any obstacle (see figure 1.4).

Presently, as claimed in [2, 3, 4, 5, 6], there are three major paradigms for autonomously controlling a robot that is intended to perform a navigation task like that of figure 1.4. Specifically, these paradigms are widely-known under the names of *deliberative*, *reactive*, and *hybrid*. Next, the essentials, as well as the strengths and weaknesses, of the afore-listed paradigms are examined.

The deliberative / sense-plan-act paradigm. In the early days of robotics, it was thought that the most effective manner to intelligently control a robot was by means of a continuous process of *sense-plan-act* (*SPA*). Concisely, such a process consisted of the following

steps: (1) collect the information provided by the robot's sensors and use these data to construct, or modify, a global model of the environment; (2) make a plan based on that internal model; and (3) send the planned actions / commands to the robot's actuators. These three steps were repeated until the robot did reach its final destination.

Through experimentation, the SPA paradigm has been shown to have some requirements to be actually successful in navigating a robot: on the one hand, its internal model of the world should be rather complete and highly accurate; on the other hand, changes in the real world should not occur more rapidly than the time needed to execute one sense-plan-act cycle.

As can be easily guessed, the above-mentioned requirements of the SPA paradigm are not necessarily satisfied in real-world environments (as is well-known, the large majority of existing environments are inherently dynamic and unstructured —and, hence, difficult to model). This fact explains why the SPA paradigm is not in widespread use today.

The reactive / sense-act paradigm. In the 1980s, Rodney Brooks, in view of the limitations that the deliberative / sense-plan-act paradigm had experienced in real-world scenarios, introduced the concept of reactive control. As literally said by Brooks, "reactive control is a technique for tightly coupling perception and action, typically in the context of motor behaviors, to produce timely robotic response in dynamic and unstructured worlds". Or in other words, reactive control advocates for using the real world as its own model; i.e., under this technique, there is a direct connection between perception and action, mediated neither by representation[2] nor by reasoning / planning[3]. That is why reactive control is also known as the *sense-act / SA* paradigm.

A control architecture that relies upon the SA paradigm is fundamentally composed by a set of independent modular components called *behaviors*, which are performed in parallel. As illustrated in figure 1.5, each behavior receives inputs from part or all the robot's sensors and sends commands to a subset of the robot's actuators. Multiple behaviors may take input from the same sensor as well as may generate commands for the same actuator. At this point, it is important to note that the latter observation leaves open the possibility that conflicting commands are sent to the same actuator. In order to solve this problem, a SA-based control architecture includes a coordination mechanism per actuator. Such a mechanism is in charge of deciding the specific command that will finally be executed by the corresponding actuator. As can be observed in figure 1.6, two distinct types of coordination mechanisms are essentially distinguished, namely *competitive* and *cooperative*. A competitive coordination mechanism makes the actuator execute one of the commands issued by the behaviors —and the rest of the commands are simply ignored. Alternatively, a cooperative coordination mechanism merges all the behavioral commands into one representing their consensus, and then gives it to the actuator for execution.

In the last three decades, a large number of approaches has been proposed following the SA paradigm[4]. By analyzing all these approaches, one can realize that there are two slightly different ways of understanding the SA paradigm, named *pure* and *non-pure*. Specifically, the term pure is used to refer to those approaches that fall under the

[2]The reliance on maintaining an internal model of the environment is eliminated
[3]The large amounts of time preparing plans are avoided
[4]In chapter 2, much of the work done in this field will be extensively reviewed

classical definition of the SA paradigm given by Brooks. As an essential requirement in this definition, recall that behaviors are only allowed to compute their output commands on the basis of the (local) information being currently gathered by the robot's sensors. Comparatively speaking, non-pure approaches are characterized by somewhat relaxing the above requirement. To be precise, in a non-pure context, behaviors can compute their commands by considering, in addition to the current local sensory information, a limited amount[5] of global information about the environment.

Through experimentation, the SA paradigm has been demonstrated to enable robots to move autonomously in real-world scenarios —that is to say, in scenarios where unknown and moving obstacles do exist. Such experimentation, however, has also served to reveal the major drawbacks of that paradigm. Moreover, it has been seen that these drawbacks are different depending on whether the SA paradigm is adopted in its either pure or non-pure form. In this respect, notice that: (1) robots that navigate according to the pure form of the SA paradigm —or equivalently said, robots that are purely reactive— are known to have difficulties for managing complex scenarios (these robots get frequently trapped in concave obstacle configurations, such as the typical U-shaped canyon); (2) robots based on the non-pure form of the SA paradigm —or in fewer words, robots that are non-purely reactive— exhibit a more intelligent way of navigation, in the sense that they are able to successfully operate in complex scenarios, although at the cost of increased memory and processing demands.

Before concluding, it should also be stressed that there is one more drawback now shared by both purely and non-purely reactive robots: as a direct consequence of the exclusive / almost-exclusive handling of local information, reactive robots do perform poorly in terms of path length —i.e. these robots move usually along quite suboptimal paths to reach its destination.

The hybrid / plan, sense-act paradigm. Many modern approaches to robotic control attempt to combine the planning capabilities of deliberative systems with the responsiveness of reactive systems. Such a combination is the essence of the so-called hybrid paradigm. While the deliberative paradigm relies on a sense-plan-act perspective and the reactive paradigm follows with sense-act, the hybrid paradigm typically takes the form of *plan*, *sense-act* (with the comma meaning parallel execution). A hybrid approach, in its basic structure, is composed by two layers, one deliberative and the other reactive, which work as follows: the reactive layer performs a set of sense-act behaviors, and the deliberative layer observes the progress of such behaviors and suggests direction based on reasoning, planning, and problem-solving.

In this section, we have provided an introductory overview of the most outstanding paradigms in robot control, including deliberative systems, reactive systems, and hybrid systems. As a general conclusion, we can state that, while each system is appropriate in selected contexts, hybrid approaches are very adaptable for accommodating a large variety of robotic control scenarios with sufficient planning and reactive capabilities.

[5] of the order of a few bytes

1.4 Objectives and Structure of the Document

The overall objective of this dissertation is to develop a set of improved reactive and deliberative / path-planning methods which can be used as basic components to build new advanced hybrid control architectures for mobile robot navigation. In this respect, it is especially important to remark that the task of building new hybrid systems with the methods here developed is considered to be outside the scope of this dissertation, and, hence, it is left for future work.

Going deeper into the above, this dissertation establishes some particular objectives to be achieved by the reactive and path-planning methods being proposed. Specifically, these objectives are:

- ◇ **Regarding the reactive methods**. Along the past years, a large number of purely reactive methods has been reported. Broadly speaking, each of these methods is known to be best-suited to perform successful navigation in different contexts: some of them are characterized by safely moving the robot through very small spaces by taking into account the shape, dynamic, and kinematic constraints of the robot, others, however, are able to navigate through outdoor rough terrains, etc. It seems thus, that one can (almost) always find a purely reactive method that suits one's needs. Nevertheless, to be fair, not all needs can actually be met, since purely reactive methods suffer from an important weakness that severely limits their application: as is well-documented, purely reactive robots get usually stuck in dead-end obstacle configurations.

  This dissertation is not focussed on proposing a specific purely reactive method with the added advantage of avoiding the aforementioned stuck situations. Our objective goes one step forward than this by defining a generic way to transform several of the current purely reactive methods into equivalent methods not having the stucking problem or, what is more, into equivalent methods able to ensure completeness.

- ◇ **Regarding the path-planning methods**. This dissertation is intended to put forward various global path-planning methods that work effectively in time-constrained scenarios, i.e. in scenarios where the time available for planning is variable and limited.

  To cope with these scenarios, we will propose methods whose quality of the planned path degrades / grows gracefully as computation time decreases / increases. Or more strictly speaking, we will propose new path-planning methods of type *anytime*.

With the preceding objectives in mind, the rest of the document is organized as follows:

Chapter 2: *State of the Art in Reactive Navigation.* This chapter surveys current approaches to reactive robot control for navigation.

Chapter 3: *Traversability and Tenacity: Two New Concepts for Improving Navigation of Purely Reactive Control Systems under Limited Sensing Capabilities.* This chapter describes a novel framework for purely reactive navigation in highly complex scenarios. This framework, concisely known as T^2, gives robots the ability to avoid getting stuck when facing troublesome obstacle configurations (or in other words, under the T^2 framework, robots turn out to be immune to the so-called *local minima* problem [7]).

The T^2 framework is ultimately conceived as a means to enhance existing purely reactive approaches; that is to say, such a framework, when applied to a current state-of-the-art technique in the field of purely reactive navigation, allows this technique to successfully carry out more complex navigation tasks than originally.

Any technique to which the T^2 framework is applied does become rid of stuck situations, but it is not actually prevented from generating endless cyclic trajectories. In short, this means that there is no guarantee that the robot will always reach its destination.

Chapter 4: *Achieving a Better Path Length Performance for the Algorithm Bug2.* This chapter succeeds in improving one of the non-purely reactive approaches with best trade-off between simplicity —in terms of computational demands— and performance —in terms of the length of the path traveled by the robot—, called *Bug2*. To be more exact, this Bug-like approach[6] is modified in such a way that: (1) all the original outstanding properties of Bug2 are maintained (we are especially referring to its simplicity as well as its capability to ensure the completion of any feasible navigation task); (2) a better performance is achieved (i.e. the robot is able to find shorter paths to its destination).

Chapter 5: *T^2-based Reactive Navigation with Global Proofs.* This chapter extends the T^2 framework of chapter 3 by giving it the property of completeness. In this way, a robot based on this framework is now proved to globally converge to its destination —whenever a path exists.

The global convergence criterion that is incorporated into the T^2 framework is inspired by how completeness is accomplished by the algorithm Bug2 (recall that this algorithm is the one that is improved in chapter 4).

Chapter 6: *The Use of Different Bug-like Strategies for Building Efficient Deterministic Anytime Path Planners.* This chapter presents two new algorithms for deterministic global path planning that are aimed at real-time domains because of their anytime nature. These algorithms, briefly named *ABUG* and *vABUG*, are constructed on the basis of distinct Bug-like strategies. To be precise, *ABUG* is based on the enhanced version of the algorithm Bug2 proposed in chapter 4, whereas *vABUG* relies on the classical strategy *VisBug*. Both planners efficiently provide a series of increasingly better paths in problems of low dimensionality, such as those planning problems typically concerned with low-cost robotics applications. Through experimentation, *ABUG* and *vABUG* are shown both to plan better paths and to perform much faster than the most popular current anytime approaches.

Chapter 7: *Conclusions and Future Work.* The last chapter concludes this dissertation by summarizing the work done and presenting all the contributions made, including all the publications arising from this Ph.D. work. To end with, some recommendations for future work are also given.

[6] As will be seen in section 2.6, Bug-like strategies do represent a family of non-purely reactive approaches mainly characterized by imitating the biological behavior of some bugs / insects

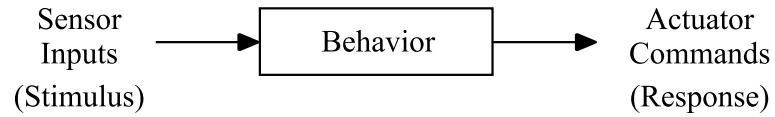


Figure 1.5: Mapping from sensor inputs to actuator commands through the behavior concept.

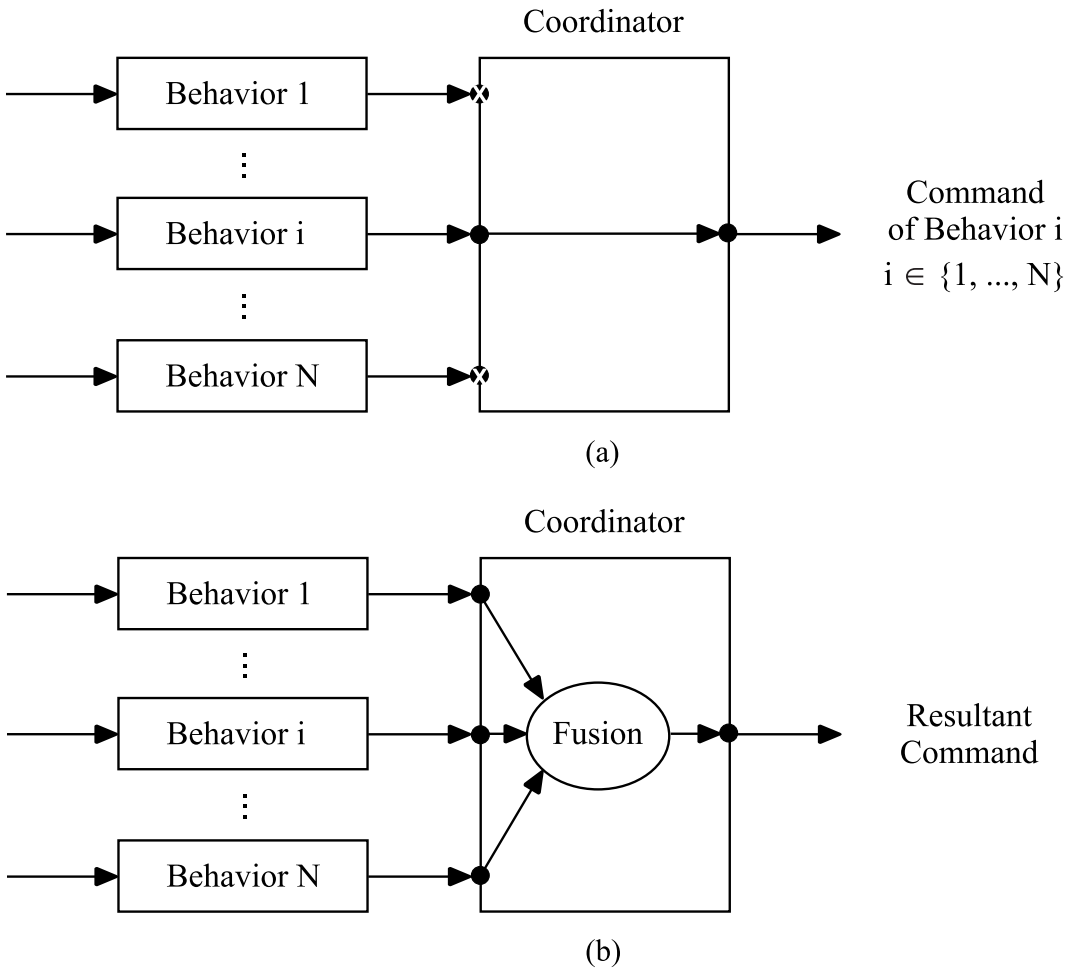


Figure 1.6: Different methodologies of coordinating behavioral commands for conflict resolution: (a) *competitive* methods; (b) *cooperative* methods.

Chapter 2

State of the Art in Reactive Navigation

This chapter is intended to give a general overview of the most popular techniques that have been devised to solve the navigation problem in a reactive way. These techniques are described after a brief introduction to the related theory, when necessary.

2.1 Artificial Potential Fields

A *potential field* is a differentiable real-valued function $U : \mathbb{R}^m \rightarrow \mathbb{R}$, whose value can be seen as energy, being hence its gradient a force. The *gradient* $\vec{\nabla}U(X)$ —X denotes a robot configuration— is a vector which points in the local direction that maximally increases U. Additionally, a *gradient vector field* assigns the gradient of some potential function to each point on the m-dimensional space.

The *Potential Fields Method (PFM)*, as originally proposed in [8], directs a robot as if it was a particle moving in a gradient vector field. Gradients can be intuitively understood as forces acting on a positively charged particle which is attracted to the negatively charged goal. Obstacles also have a positive charge which forms a repulsive force field around them causing the robot to go away. The combination of repulsive and attractive forces is expected to guide the vehicle from the starting location to the target while avoiding obstacles.

By way of example, assuming that $m = 2$ and X in $U(X)$ represents the robot's position on the corresponding plane $\left(\mathbb{R}^2\right)$, figures 2.1(a) and (b) depict a typical gradient vector field for, respectively, the attractive and the repulsive potential functions. Moreover, the superposition of both fields is illustrated in figure 2.2 together with a possible trajectory of the robot.

Figure 2.3 shows the block diagram of a reactive control system as suggested by the original / classical potential fields approach. As can be observed, the control architecture consists of two behaviors named *GoTo* and *AvoidObstacles*, which are responsible for, respectively, generating the attractive and the repulsive forces at the current robot's location[1] on the basis of the local sensory information. Besides, a cooperative coordination mechanism based on vector addition is used in order to merge the behavioral responses. Notice that the addition is weighted according to *gains* G_i, which encode the relative strength of each behavior.

[1] For a faster processing, the entire potential fields are never computed. Only each field's contribution at the instantaneous position where the robot is currently located is calculated. The whole potential field is simply represented for the reader's edification

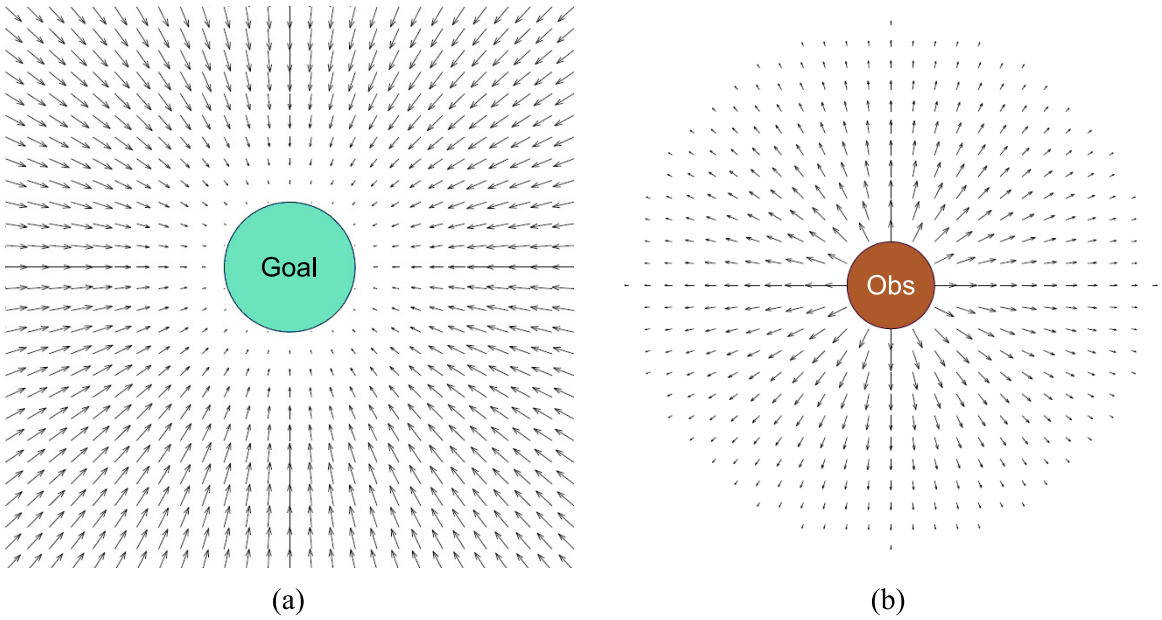


Figure 2.1: The attractor-repeller paradigm: potential fields for a goal (a) and an obstacle (b).

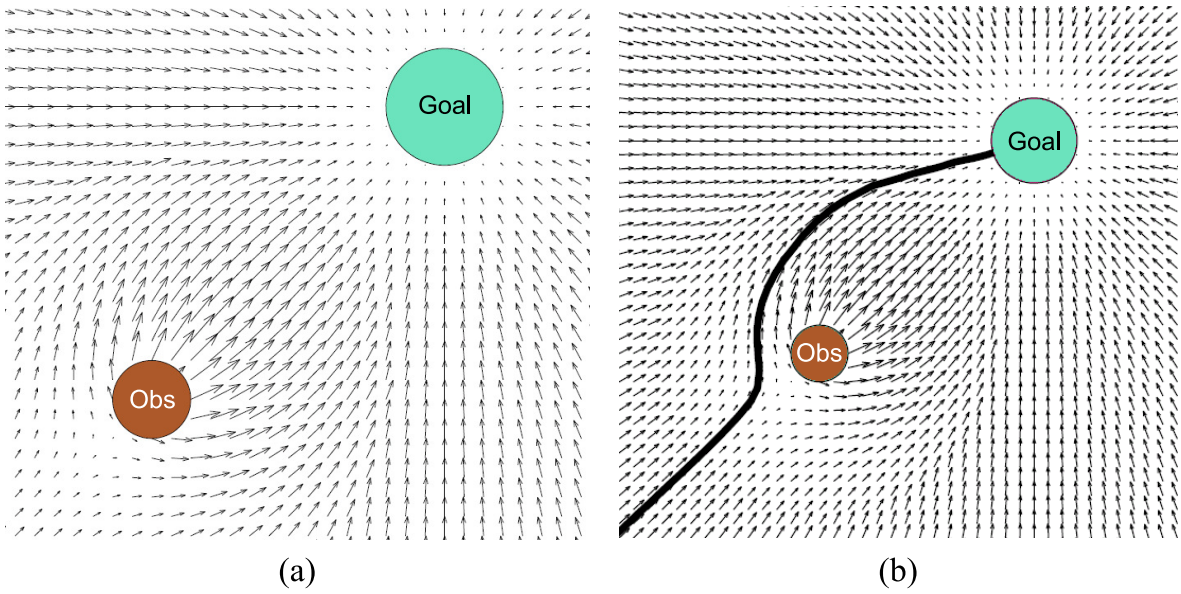


Figure 2.2: (a) linear combination of the potential fields depicted in figure 2.1, with (b) exemplifying a path for a robot moving within this simple scenario.

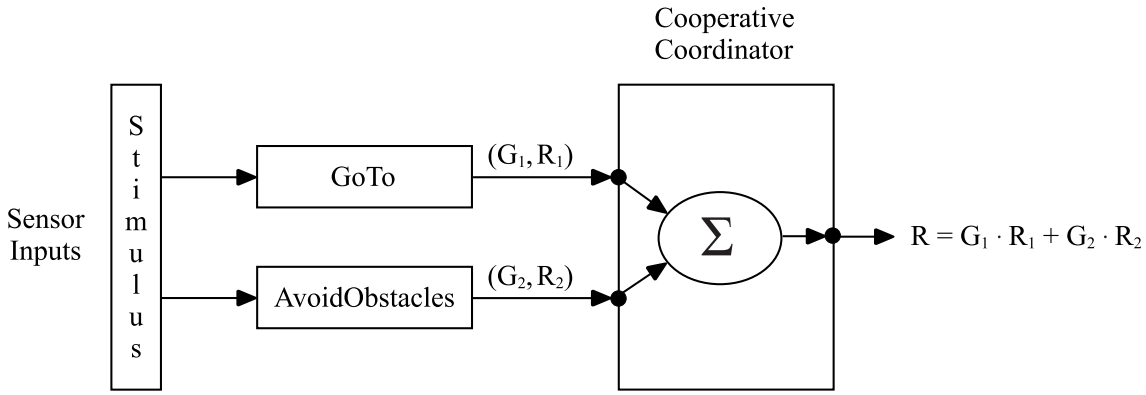


Figure 2.3: Scheme of the classical potential fields approach.

Next, let us finish the introduction to the basic principles of potential fields by remarking the most important shortcomings that are inherent[2] to the approach (see [9] for a further explanation):

1. Possible existence of *local minima* in the resultant potential field, where the robot may get indefinitely stuck. The potential fields approach can be imagined as a gradient descent method which ensures the convergence to a minimum in the field. Unfortunately, there is no guarantee that this minimum is the global minimum located at the goal point.

2. Lack of an oscillation-free motion when the vehicle navigates among very close obstacles at high speed.

3. Impossibility to go through small openings.

Among the above shortcomings, the first one is the best-known and most-often cited problem with *PFMs*, limiting the applicability of the strategy to straightforward navigation tasks. In that respect, figures 2.4 (a) and (b) show two simple scenarios where such a problem does arise, preventing thus the robot from reaching its target. As can be seen, in the former scenario, the vehicle is initially attracted to the goal as it approaches the U-shaped obstacle. The goal continues attracting the robot, but the lower wall of the obstacle deflects the vehicle upwards until the local detection of the upper wall begins also to influence the robot's path. At this moment, the combined effect of the lower wall and the upper wall of the obstacle keeps the robot halfway between these walls. The vehicle continues making progress towards its target until it, ultimately, achieves a position in the environment where the effect of the obstacles counteracts the attraction of the goal. In other words, the robot has got to a point q where $\| \vec{\nabla} U(q) \| = 0$, not being q the goal. The same occurs in figure 2.4 (b) where the repulsive force balances out the attractive one.

In the rest of the section, several *PFM*-based navigation strategies are deeply discussed. As expected, they try to alleviate some of the above-listed shortcomings.

[2]These problems are independent of the particular implementation

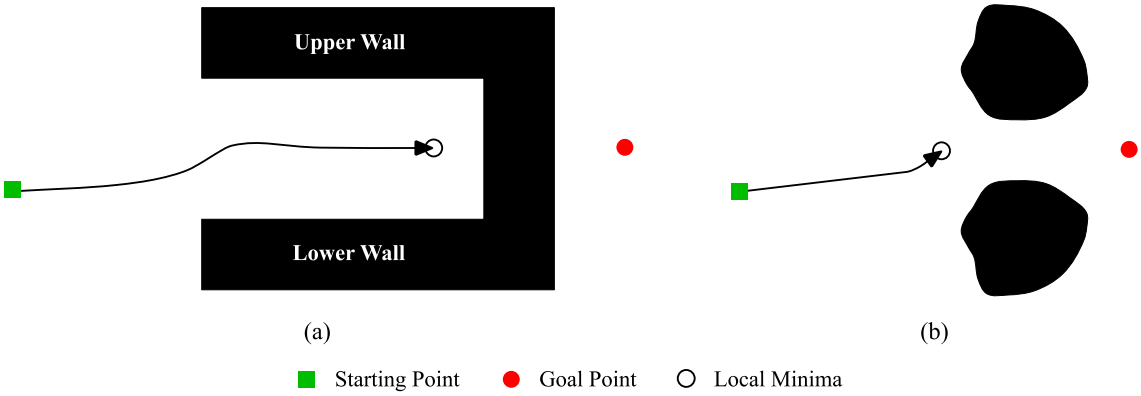


Figure 2.4: The *local minima* problem with concave (a) and convex (b) obstacles.

2.1.1 The Generalized Potential Fields Method

Author(s): Bruce KROGH [10]

Reference(s): [10] 1984

Description: The magnitude of a repulsive force ($F_{r,o}$) calculated by the classical potential fields approach [8] exclusively depends on the distance to the corresponding obstacle (d_o). Nevertheless, the *Generalized Potential Fields Method (GPFM)* additionally considers the relative velocity between the robot and the obstacle in the computation of such a magnitude. More exactly, the repulsive potential field is defined to be inversely proportional to the so-called avoidance period, which means the time difference between the minimum deceleration of the vehicle on the remaining distance to the obstacle following the current direction of motion, and the maximum deceleration. Equation 2.1 formalizes this idea, where α represents the maximum deceleration of the robot and v the velocity component in the obstacle direction. As can be observed, an infinite repulsive force is generated when the estimated time to collision coincides with the minimum time required to stop the vehicle.

$$\| F_{r,o} \| = \frac{\alpha v}{2 d_o \alpha - v^2}. \tag{2.1}$$

This approach was combined with a global planner in [11].

2.1.2 The Virtual Force Field

Author(s): Johann BORENSTEIN [12] and Yorem KOREN [12]

Reference(s): [12] 1989

Description: The novelty of this approach, entitled the *Virtual Force Field (VFF)*, lies in the integration of two already-known concepts: "Certainty Grids" [13, 14] for obstacle

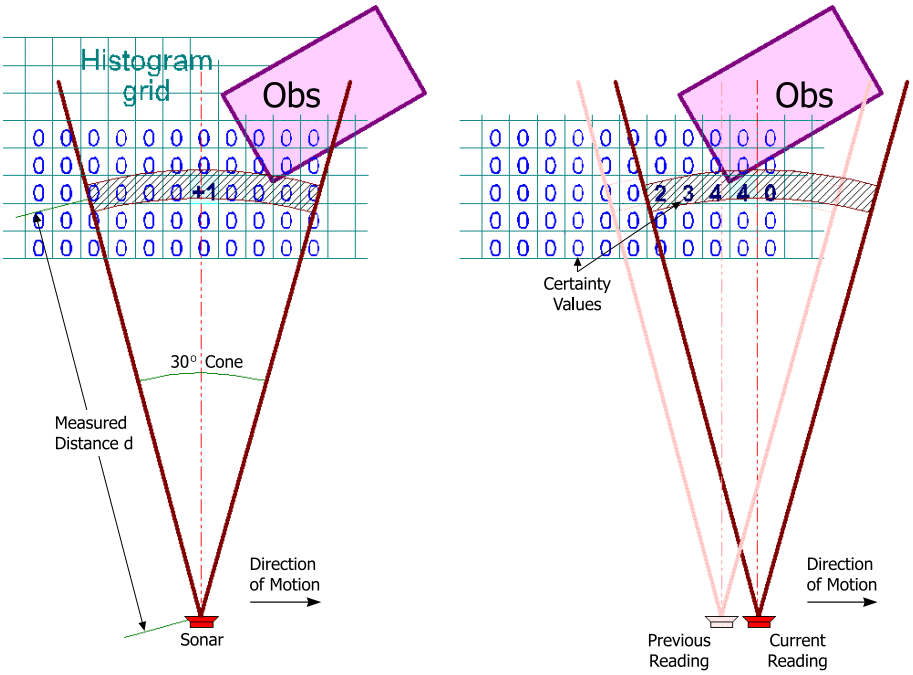


Figure 2.5: The histogram grid world model.

representation, and "Potential Fields" for navigation. More precisely, the *VFF* method works as follows:

1. A histogram grid (C) is updated by firing 24 ultrasonic sensors distributed around the robot. It is specifically done in accordance with the next two steps exemplified in figure 2.5:
   - For each range reading, the cell that lies on the acoustic axis and corresponds to the measured distance d is augmented, increasing thus the certainty value (CV) of the cell.
   - A histogramic pseudo-probability distribution is obtained by a continuous and rapid sampling of the sensors while the robot is moving.
2. A virtual window moves with the vehicle and overlays a square region of the histogram grid. The cells which are covered by this window are called active cells.
3. Each active cell exerts a virtual repulsive force $F_{r,c[i,j]}$ on the robot. The magnitude of this force is proportional to the CV of the cell, and inversely proportional to r^2, where r denotes the Euclidean distance between the cell and the vehicle.
4. Next, all virtual repulsive forces generated from the active cells are added up to yield the resultant repulsive force vector F_r.
5. A constant-magnitude virtual attractive force F_a is also applied to the robot by the target. The addition of F_r and F_a produces the final force vector F. Afterwards, the steering of the vehicle is aligned with F to avoid the obstacle.

This strategy has demonstrated to be specially well-suited for the accommodation of inaccurate sensor data as well as for sensor fusion, enabling, at the same time, the continuous motion of the robot without stopping in front of obstacles.

2.1.3 The Vector Field Histogram

Author(s): Johann BORENSTEIN [15, 16, 17], Yorem KOREN [15] and Iwan ULRICH [16, 17]

Reference(s): [15] 1991, [16] 1998, [17] 2000

Description: The strategy *VFF* described in section 2.1.2 is known to suffer from three main problems which are briefly pointed out in the following:

- Difficulty for generating smooth trajectories (drastic changes between two successive responses may occur).
- Impossibility of passing among closely spaced obstacles such as doorways.
- Tendency of the robot to oscillate in narrow corridors.

The *Vector Field Histogram (VFH)* [15] is, precisely, an approach which was developed with the intention of remedying these shortcomings based on the next observation: carefully analyzing the aforementioned problems, it was concluded that they were caused by the excessive data reduction occurring when the individual repulsive forces from the histogram grid cells $\left(F_{r,c[i,j]}\right)$ were added up to compute the resultant force vector (F_r). Notice that, in this way, the information about the local distribution of the obstacles is mostly lost.

VFH extends the *VFF* method by using an intermediate data structure known as the polar histogram (H), which is essentially an array composed of 72 angular sectors 5-degree wide. In short, the strategy involves the next two steps:

1. A window moves with the robot overlaying a square region of cells in the histogram grid. The content of each active cell is mapped into the corresponding angular sector of the polar histogram, as exemplified in figure 2.6(a). As a result, each sector k holds a value h_k representing the obstacle density in the range of directions which consists of.
2. Later, a threshold on the polar histogram determines the candidate directions of motion $\Big($look at figures 2.6(b) and (c)$\Big)$. Finally, among all the candidates, the algorithm selects the one that most closely matches the direction to the target.

Some improvements to the *VFH* strategy can be found in [16] (VFH^+) and [17] (VFH^*). Concisely, the former enhances the original method in several aspects which result in a greater reliability, an easier parameter tuning, and the generation of smoother paths. On the other hand, VFH^* abandons the reactive character of the approach by using a planner based on an A* algorithm, which verifies that a particular candidate direction guides the robot around an obstacle.

2.1.4 Motor Schemas

Schema theory appeared in the 18th century as a model motivated by the biological sciences for the explanation of the behavior as well as the mechanisms of memory and learning. In 1981,

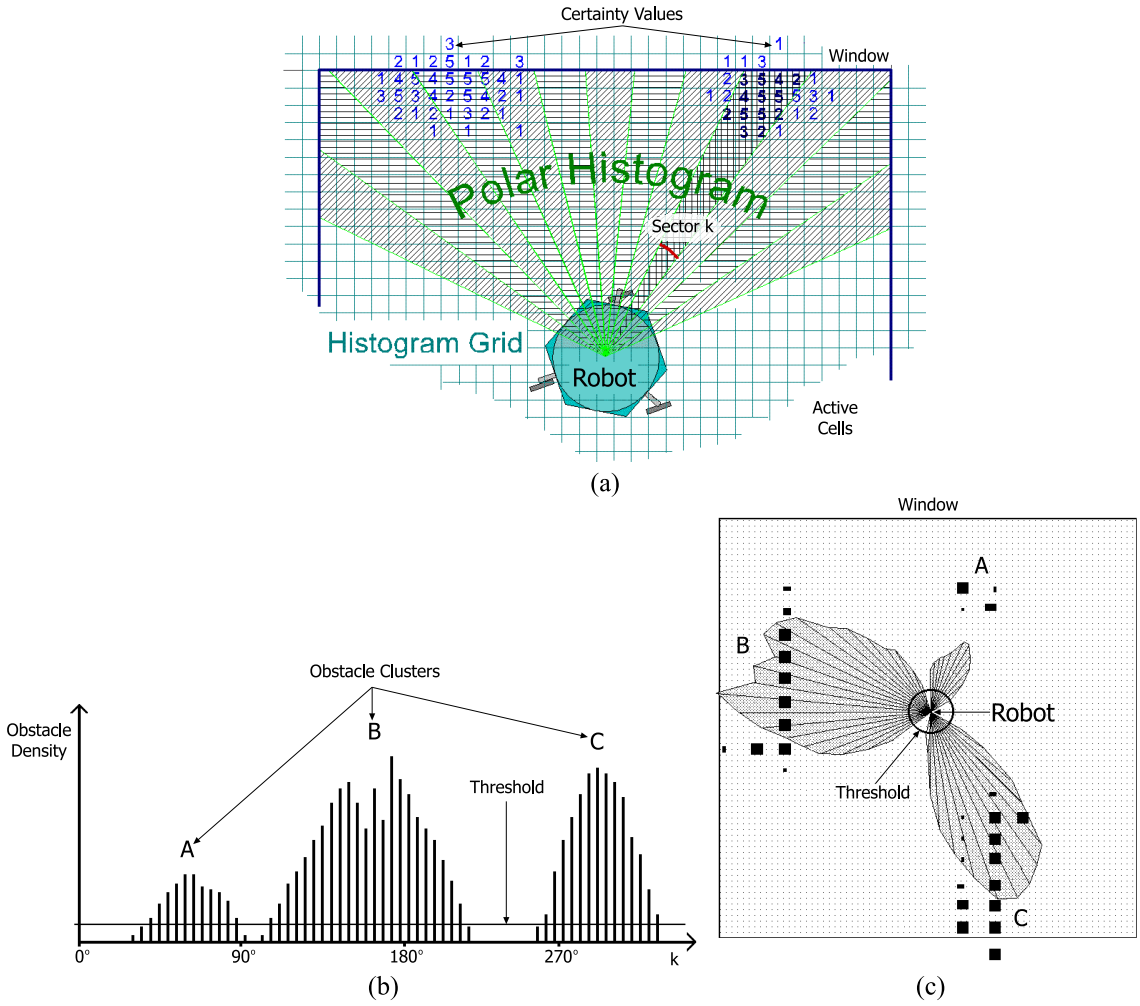


Figure 2.6: The polar histogram: (a) mapping of active cells; (b) histogram building for the scenario depicted in (c).

Michael Arbib adapted for the first time such theory to a robotic system. To be more exact, a simple schema-based model inspired by the behavior of a toad was put forward to control a robot [18]. Since then, numerous methodologies have been proposed for both specifying and designing behavioral / reactive control architectures based on schema theory. In that respect, one of the most outstanding proposals is named *Motor Schemas (MS)*. It was suggested in [19] by Ronald Arkin and differs from other behavioral approaches in several significant ways, which are briefly summarized next:

- Behavioral responses are all given in a single uniform format, namely vectors generated using potential fields.
- Coordination is accomplished through cooperative means by vector addition.
- Behaviors can be either instantiated or deinstantiated at any time on the basis of perceptual events.
- Each behavior can contribute in varying degrees to the robot's overall response according to the concept of *gain*.

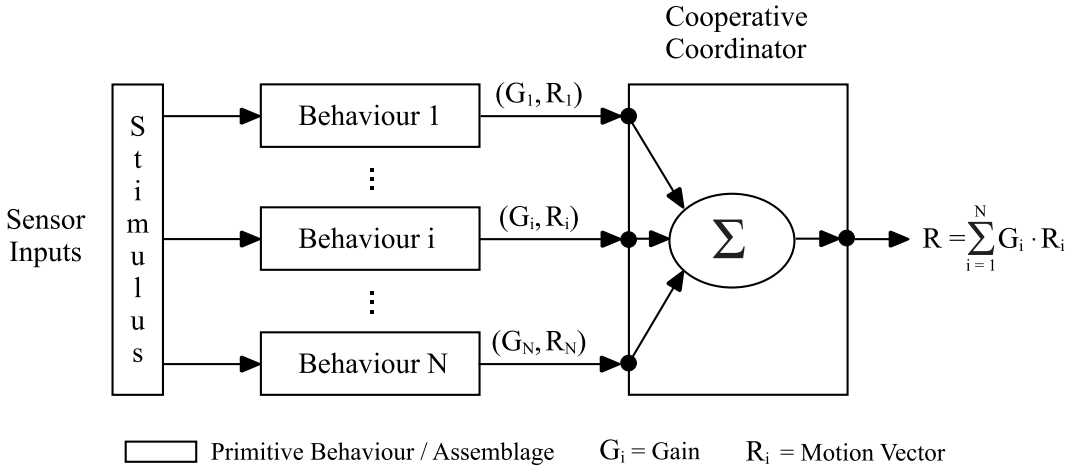


Figure 2.7: A schema-based control system.

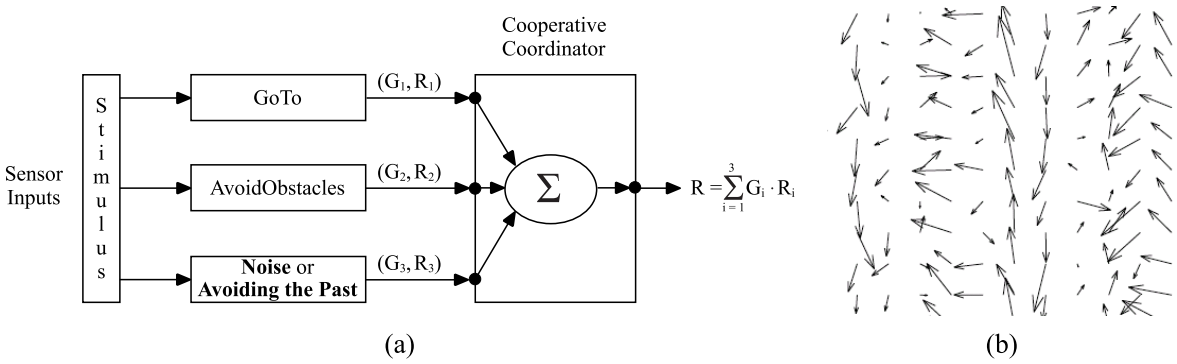


Figure 2.8: The *Noise* and the *Avoiding the Past* motor schemas: (a) integration into a basic navigational control system; (b) random potential field linked to the former.

- Abstraction can be used to construct high-level behaviors from simpler ones by taking advantage of the so-called *behavioral assemblages*. Fundamentally, an assemblage is a package composed by a set of primitive behaviors and, optionally, other assemblages, which are cooperatively coordinated.

Figure 2.7 illustrates the generic structure of a schema-based control system. At this point, it is important to note that, like the classical potential fields approach [8], *MS* is not immune to the local minima problem. In the following, two different strategies called *Noise* and *Avoiding the Past* are discussed, trying to address this problem. In short, both of them consist in incorporating a specific *motor schema* / behavior, named as the corresponding strategy, into a control architecture composed by the GoTo and AvoidObstacles behaviors $\Big($look at figure 2.8(a)$\Big)$.

2.1.4.1 Noise

Author(s): Ronald ARKIN [1]

Reference(s): [1] 1998

Description: Injecting randomness into a schema-based control system is a technique used to deal with local minima. To this end, a *Noise*-type behavior is simply added to the control architecture. This behavior generates a random potential field like the one depicted in figure 2.8(b), which helps to ensure the robot's progress.

2.1.4.2 Avoiding the Past

Author(s): Ronald ARKIN [20] and Tucker BALCH [20]

Reference(s): [20] 1993

Description: The *Avoiding the Past* motor schema repels the robot from locations which were already visited. With this purpose, a local map of the environment implemented as a two-dimensional grid is stored in memory, where a different value is assigned to visited and non-visited locations. As the robot visits an area more times, the values of the corresponding cells in the grid increase and, consequently, the resultant repulsive force exerted by such cells increases as well. In this way, it is intended to favor the continuous exploration of new regions of the navigation environment, avoiding thus, at least apparently, the robot gets stuck into a local minimum. See algorithm 2.1 for further details.

2.1.5 Micronavigation

Author(s): Alessandro SCALZO [21], Antonio SGORBISSA [22, 21, 23] and Renato ZACCARIA [21, 23]

Reference(s): [22] 2000, [21] 2003, [23] 2004

Description: This approach named *Micronavigation (μNAV)* tries to solve the problem of autonomous robot navigation from a minimalist point of view. Based on artificial potential fields, μNAV uses a handful of bytes for generating a path to the goal. Successful results are reported in complex scenarios such as maze-like environments. Nevertheless, it is important to stress that the mission completion cannot be always guaranteed.

Specifically, the motion law of the robot at position x is defined according to equation 2.2, where v_{ref} is a reference value for the speed, $U_r(x)$ denotes the repulsive potential field, w_g, w_f and w_t represent three weighting functions and, finally, g, f and t are three unit vectors oriented as shown in figure 2.9. As can be observed, g points towards the target, while f corresponds to the sum of the repulsive forces exerted by the obstacles, and the direction of t is tangential to the equipotential line passing through x.

Algorithm 2.1 Computation of the output vector for *Avoiding the Past*

{Initial settings for the component vectors X —XVec— and Y —YVec— as well as a global counter of the number of visits}
XVec.mag = 0;
XVec.dir = 90;
YVec.mag = 0;
YVec.dir = 0;
Visits = 0;

{Computing the magnitude of the component vectors based on the robot's visits to a local square region centered in RobotPos —the current robot's position— and whose size is given by the PastHorizon parameter. M represents the up-to-date 2D map of visited locations (accordingly, M[i,j] denotes the precise number of visits of the robot to the cell ij)}
for i = (RobotPos.X − PastHorizon) to (RobotPos.X + PastHorizon) **do**
  for j = (RobotPos.Y − PastHorizon) to (RobotPos.Y + PastHorizon) **do**
    if i < RobotPos.X **then**
      XVec.mag = XVec.mag + M[i,j];
    else if i > RobotPos.X **then**
      XVec.mag = XVec.mag − M[i,j];
    end if
    if j < RobotPos.Y **then**
      YVec.mag = YVec.mag + M[i,j];
    else if j > RobotPos.Y **then**
      YVec.mag = YVec.mag − M[i,j];
    end if
    Visits = Visits + M[i,j];
  end for
end for

{Obtaining the direction of the output vector from XVec and YVec}
TempVec = SumVector(XVec, YVec);
PastVec.dir = TempVec.dir;

{Determining the magnitude of the output vector in accordance with the total number of visits carried out to the aforesaid local region}
$$\text{PastVec.mag} = \text{PastGain} * \left(\text{Visits} \;/\; \left((2 * \text{PastHorizon} + 1)^2 * \text{MaxVisitsPerCell} \right) \right);$$

return PastVec;

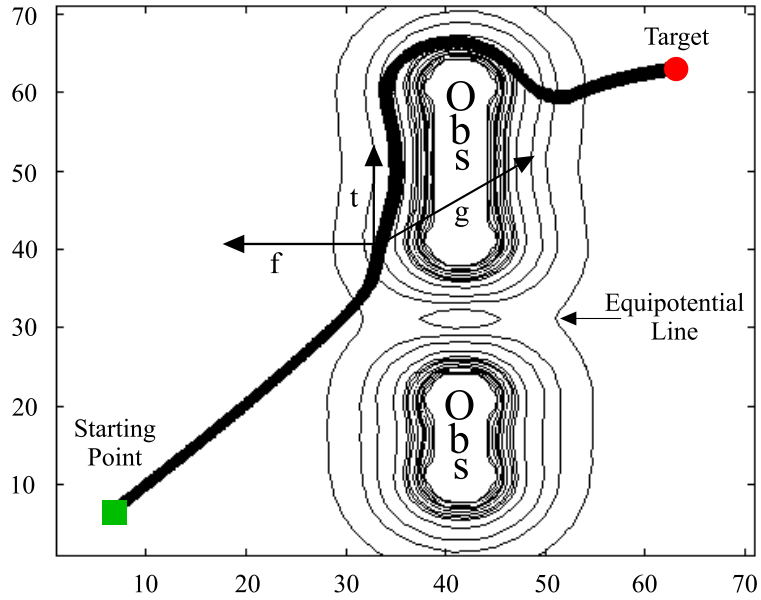


Figure 2.9: Robot's trajectory of the μNAV algorithm in a simulated environment.

$$v(x) = \Big(w_g(U_r(x)) \cdot g(x) + w_t(U_r(x)) \cdot t(x) + w_f(U_r(x)) \cdot f(x)\Big) \cdot v_{ref}. \tag{2.2}$$

Different behaviors can be obtained by establishing different weighting functions in equation 2.2. This fact is exploited by μNAV to provide the robot with a hierarchy of five simple behaviors designed for smooth obstacle avoidance, and for detecting and escaping from deadlock situations.

2.1.6 Harmonic Potential Fields

Author(s): Brian BURNS [24], Chris CONNOLLY [24], Daniel KODITSCHEK [25], Elon RIMON [25], Richard WEISS [24]

Reference(s): [24] 1990, [25] 1992

Description: A *Harmonic Potential Field (HPF)* refers to a special type of artificial potential function whose generation is constrained by the use of *Laplace*'s equation. As a main feature, these functions do not exhibit spurious local minima, as opposed to *PFMs*. However, this advantage is accomplished at the cost of greatly increasing the computational demands, which turns *HPF* into a less performance-oriented / less-suited method for reactive navigation. Finally, it is important to note that, as claimed by the authors, the computational drawback of *HPF* can be alleviated by exploiting the intrinsically-parallel formulation of *Laplace*'s equation. In this way, a solution to this equation is expected to be quickly obtained by operating with several processors.

2.2 Learning of Behavioral Parameters

2.2.1 GA-Robot

Author(s): Ronald ARKIN [26, 27], Gary BOONE [27], Michael PEARCE [26, 27] and Ashwin RAM [26, 27]

Reference(s): [26] 1992, [27] 1994

Description: This strategy called *GA-Robot* explores the application of *genetic algorithms (GA)* to the off-line and unsupervised learning of the behavioral parameters of a reactive control architecture made up of three motor schemas: GoTo, AvoidObstacles, and Noise. The method is used for reducing the effort required to configure such navigation system by learning parameter settings that optimise performance metrics of interest such as safety, speed or distance in various kinds of environments, named *ecological niches* by its authors. The resultant sets of parameters can be applied to similar scenarios which were not presented in the learning phase. The approach has been exclusively evaluated through computer simulations.

2.2.2 Learning Momentum

Author(s): Ronald ARKIN [28, 29, 30], Russell CLARK [28], James LEE [29, 30] and Ashwin RAM [28]

Reference(s): [28] 1992, [29] 2001, [30] 2003

Description: The approach *Learning Momentum (LM)* can be considered a crude form of reinforcement learning, where, if the robot is working well, it keeps doing the same and even a bit harder. Conversely, if the vehicle is not working properly, it tries something different. With this purpose, two tasks are sequentially carried out: on the one hand, the identification of the robot's performance and, on the other hand, the adaptation of the robot's behavior according to both such performance and the current environmental conditions. As for the former task, *LM* relies on recent experience and a set of heuristic rules for determining when good progress to the goal is being made. The alteration of the vehicle's behavior, nevertheless, is performed by altering the gain values as well as other parameters of a reactive control system composed by three motor schemas: GoTo, AvoidObstacles, and Noise.

LM can be actually implemented in two different ways named *ballooning* and *squeezing*. Specifically, these strategies consist of, respectively, increasing / decreasing the sphere of influence (SOI) around the robot[3] when it makes little or no progress towards the target. In short, ballooning works better when facing obstacles such as box-shaped canyons, while squeezing allows the vehicle to suitably navigate through environments built with small and closely spaced obstacles. The choice between both options must be taken in an off-line manner.

Algorithm 2.2 describes the *LM* approach in pseudocode.

[3]The SOI is a region that limits the influence of the obstacles on the robot motion to those located inside

Algorithm 2.2 *Learning Momentum*

```
{A call to this routine is inserted at the end of the robot's cycle. On the other hand,
the value of Steps is initially zero}

{Nothing is done until the robot has taken a certain number of steps}
Steps = Steps + 1;
if Steps == HistoryInterval then
  Steps = 0;

  {Updating some relevant information from the data collected during the last HistoryIn-
  terval steps}
  Calculate the average movement and progress of the robot as well as the number of
  obstacles which have been detected

  {Identifying the current situation}
  if AverageMovement < M then
    Situation = NoMovement;
  else if AverageProgress > P then
    Situation = Progress;
  else if ObstacleCount > O then
    Situation = NoProgressWithObstacles;
  else
    Situation = NoProgressWithoutObstacles;
  end if

  {Altering the behavioral parameters}
  for each behavioral parameter do
    Modify its value by adding it a number selected in a random way within a specific range,
    which depends on both the parameter and the given current situation
  end for
end if
```

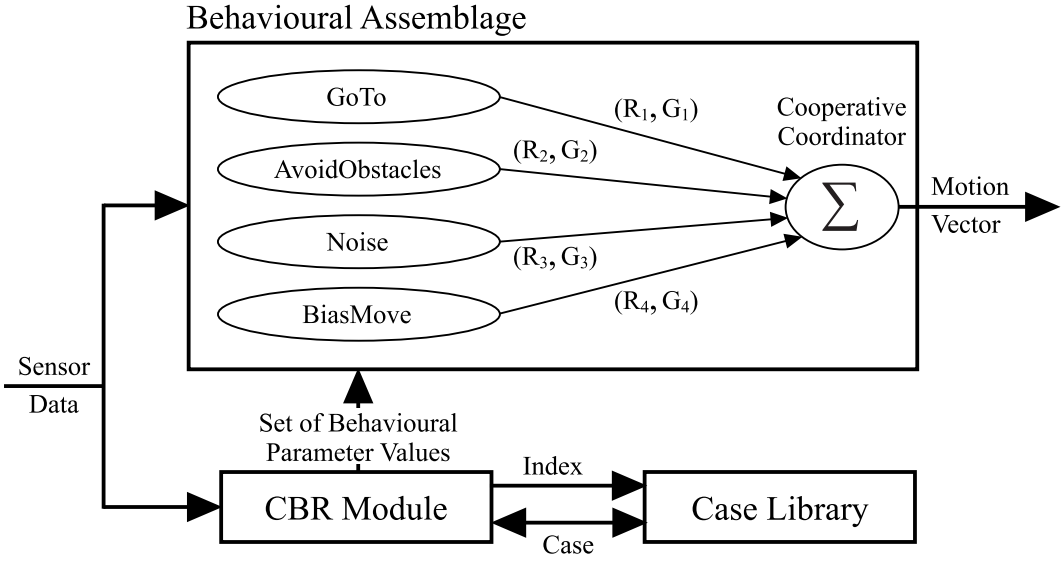


Figure 2.10: Interaction between the reactive control system and the *CBR* unit. Each behavior supplies to the coordination mechanism, in addition to its vectorial response R_i, the multiplicative gain G_i currently assigned by the *CBR* module.

2.2.3 Case-based Navigation

Author(s): Ronald ARKIN [31, 32, 33, 34, 35], Russell CLARK [31], Michael KAESS [33], Zsolt KIRA [35], James LEE [34], Maxim LIKHACHEV [32, 33, 34], Kenneth MOORMAN [31] and Ashwin RAM [31]

Reference(s): [31] 1997, [32] 2001, [33, 34] 2002, [35] 2004

Description: These papers put forward, in a progressive way, a complex strategy, based on the use of *Case-Based Reasoning (CBR)*, for conducting autonomous navigation tasks by selecting and learning optimal behavioral parameterizations at runtime. More precisely, the proposal consists in the integration of a *CBR* unit into a specific schema-based control system, just as shown in figure 2.10. Such a unit continually chooses, from a library of cases, the set of behavioral parameter values that is best suited to the current environmental situation. After its application, the performance of the selected set — or case, from now on— is assessed in terms of the progress achieved towards the robot's target. What is more, if the analysis of performance concludes that no progress or little progress has been made, the case is revised so as to improve the navigation results to be obtained in future applications of that case.

2.3 Fuzzy Logic Control Systems

Many fuzzy logic control systems have been proposed in the framework of mobile robot navigation. They possess an inherent skill in managing uncertain and imprecise information using linguistic rules, which are defined on the basis of human knowledge and experience. The generic block diagram of a fuzzy controller is depicted in figure 2.11. As can be observed,

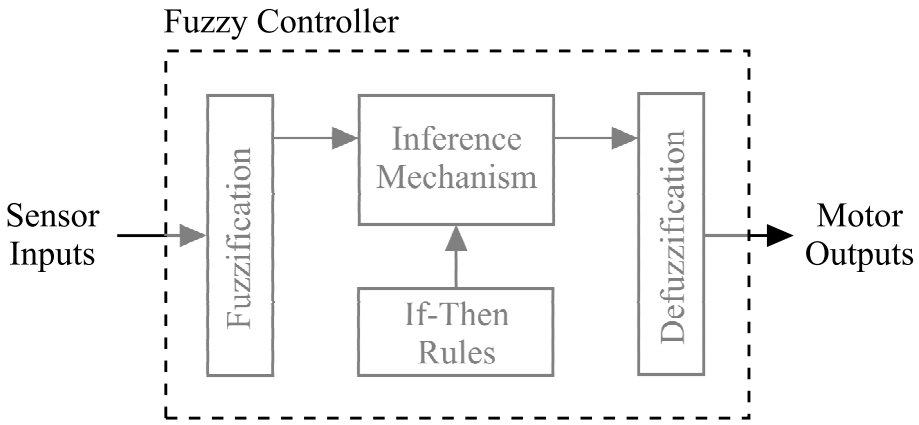


Figure 2.11: Main components of a fuzzy logic control system.

first of all, sensor signals are translated into linguistic values in the context of the so-called *fuzzification* process. Afterwards, the fuzzy inference step takes place evaluating the set of if-then rules which describes the system's behaviors. Besides, this module also identifies the most suitable rules to be applied according to the current situation and computes the values of the output linguistic variables. Finally, a process named *defuzzification* is executed which converts the linguistic values into real values such as, for instance, velocities / accelerations of the robot's wheels.

The pieces of work presented in [36, 37, 38] are three representative examples of fuzzy-based reactive control systems. All of them address the navigation problem in a quite similar way. Essentially, the inference mechanism of these controllers is made up of rules defining two basic behaviors: GoTo and AvoidObstacles. On the other hand, an additional module is incorporated into the system for identifying trapping situations caused by the local minima problem. Such an identification is based on recognizing the repeated traversal of the robot through the same environment by recollecting some real landmarks. Once the vehicle is known to be trapped, a ContourFollowing behavior is activated to escape from the trapping area.

2.4 Restricted Optimisation in the Velocity Space

2.4.1 The Dynamic Window Approach

Author(s): Oliver BROCK [39], Wolfram BURGARD [40, 41, 42], Dieter FOX [40, 41], Oussama KHATIB [39], Naomi LEONARD [43, 44], Petter OGREN [43, 44], Cyrill STACHNISS [42] and Sebastian THRUN [40, 41]

Reference(s): [40] 1995, [41] 1997, [39] 1999, [42, 43] 2002, [44] 2005

Description: The *Dynamic Window Approach (DWA)* [40, 41] is a popular strategy for reactive collision avoidance that considers both the kinematic and dynamic constraints of a synchro-drive robot[4].

[4]This technique can also be applied to differentially steered robots, and many non-holonomic vehicles

Kinematic constraints are taken into account by directly searching the velocity space consisting of the set of tuples (v,w) of translational v and rotational w velocities which are achievable by the vehicle. As a side note, each pair (v,w) defines a circular trajectory.

Among all velocity tuples, those that allow the robot to come to a stop before hitting an obstacle are selected, given the current position and the maximum deceleration of the vehicle. The resultant set of velocities is generically called *admissible*.

Afterwards, a dynamic window is defined restricting the admissible velocities to those that can be reached within a short time interval, given the current velocity and, once again, the maximum acceleration / deceleration capabilities of the robot. Notice that these capabilities for translation and steering are independent in a synchro-drive vehicle, which results in a dynamic window with rectangular shape.

Finally, in order to determine the next motion command, among all admissible velocities within the aforementioned dynamic window, the one that maximises the progress towards the target, the distance to the obstacles as well as the forward velocity of the robot is chosen.

Some enhancements to *DWA* can be found in [39, 42, 43, 44]. Essentially, these approaches integrate different forms of path-planning into the reactive collision avoidance technique so as to achieve global convergence.

2.4.2 The Curvature-Velocity Method

Author(s): Javier BENAYAS [45], Amador DIEGUEZ [45], Joaquin FERNANDEZ [45], Nak KO [46], Rafael SANZ [45] and Reid SIMMONS [47, 46]

Reference(s): [47] 1996, [46] 1998, [45] 2004

Description: The *Curvature-Velocity Method (CVM)* [47] is a strategy which was developed independently of *DWA*. Both approaches, however, fundamentally share the same principles. In short, *CVM*, under the assumption that the robot travels along arc of circles, finds a point in the velocity space $\langle v,\ w\rangle$ satisfying some physical and environmental constraints, and maximizing an objective function that trades off speed, safety, and goal-directness. As compared to *DWA*, *CVM* does have the advantage of not requiring the discretization of the velocity space; that is to say, in *CVM*, the search is performed over a continuous $\langle v,\ w\rangle$ space.

Some improvements to *CVM* can be found in [46] (the *Lane-Curvature Method*, *LCM*) and [45] (the *Beam-Curvature Method*, *BCM*). Specifically, these methods generate motion commands in two separate steps. In the first one, a desired goal heading is determined, whose direction can be different to that of the target point. Afterwards, the best steering command yielding a motion in the desired local direction is obtained by applying the original *CVM* approach.

2.5 The Nearness-Diagram Navigation Method, and Some Related Extensions

2.5.1 The Nearness-Diagram Navigation Method

Author(s): Javier MINGUEZ [48, 49], Luis MONTANO [48, 49] and Javier OSUNA [49]

Reference(s): [48, 49] 2004

Description: The *Nearness Diagram method (ND)* addresses navigation in dense, complex, and difficult scenarios by adopting a divide and conquer strategy based on situations. In essence, this approach relies on a complete set of mutually exclusive situations that describe the relative state of some relevant entities of the navigation problem such as, for instance, the position of both the robot and the target, the distribution and the proximity of the obstacles, and the location and the width —just differentiating between wide and narrow— of all obstacle-free / navigable regions[5]. At each time step, *ND* uses the available sensory information to identify the current situation of the robot within the above-mentioned set, and then the motion law —action— linked to the situation that has been finally recognized is suitably performed. Generally speaking, actions try to avoid the most dangerous obstacles around the robot, while keeping moving towards the target point.

See figure 2.12 for a better understanding of how *ND* works. This figure depicts the precise action which is taken by the method in a low-safety situation where several obstacles are found very close to the robot —inside the so-called security zone.

2.5.2 The Obstacle-Restriction Method

Author(s): Javier MINGUEZ [50]

Reference(s): [50] 2005

Description: The *Obstacle-Restriction Method (ORM)* arises with the purpose of achieving better paths / results than the strategy *ND* (see section 2.5.1) when navigating in open spaces. *ORM*, like *ND*, is well-suited to safely move a robot to a given location — the target— in a troublesome scenario, typically characterized by complex and cluttered distributions of obstacles. Nevertheless, in face of simpler environments with only a few sparse obstacles, *ORM* produces shorter and more natural trajectories than *ND*. To be precise, in the latter kind of environments, the enhanced behavior of *ORM* comes from computing the final action / motion command by taking into consideration all the obstacle information that is locally available for the robot —and not just a part of it, as it is put forward by *ND*, which restricts the motion of the robot based on exclusively the two obstacles that define the so-called *free-walking area*[6].

[5] These regions are also known as *gaps*

[6] As evidenced in figure 2.12, the *free-walking area* refers to the navigable region that is closest to the target

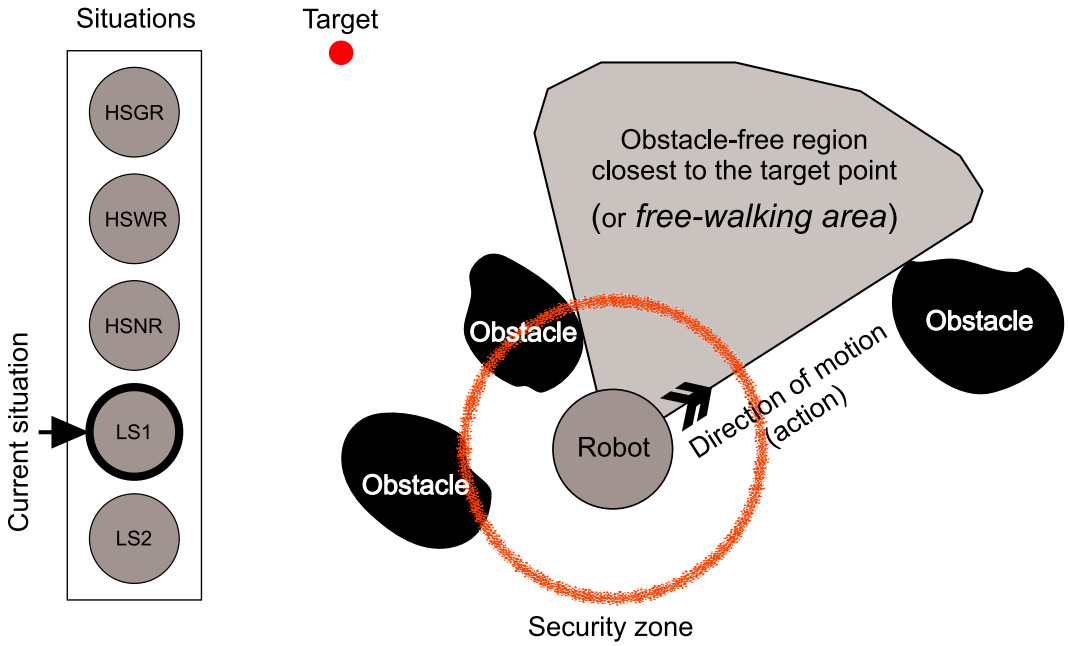


Figure 2.12: Illustrating with an example the situation / action scheme employed by *ND* to cope with the navigation problem.

At each iteration of the control cycle, *ORM* tackles the problem of obstacle avoidance / navigation in the two following steps:

1. There is a local procedure that calculates a set of subgoals —$X = \{x_0, \ldots, x_n\}$— according to the distribution of the obstacles in the scene, and the current location of the robot —x_{robot}— and the target —x_0. Later, the resultant set X is filtered by removing those subgoals whose achievement inevitably involves a collision because the robot does not fit in their corresponding passages. To this end, *ORM* builds the configuration space (C-space) by growing all the detected obstacles by the shape of the robot, which is supposed to be circular with radius R (notice that the robot is also assumed to consist of a holonomic / omnidirectional base). After this growing process, each subgoal $x_i \in X$ is checked against the existence of a non-blocked path connecting x_{robot} to x_i. In case of success, x_i is added to the new set X_f. Otherwise, x_i is discarded. *ORM* finishes its first step by choosing, from X_f, the closest subgoal to the target.

   Figures 2.13(a) and (b) highlight, in a scenario containing three main obstacles, the tasks linked to the subgoal-selector procedure described above.

2. Let $x_{closest}$ be the subgoal provided by step 1. In this second step, *ORM* determines θ_{sol}, i.e. the most promising collision-free direction of motion towards $x_{closest}$. By way of example, figure 2.13(c) shows the computation of θ_{sol} for the same scenario of figures 2.13(a,b).

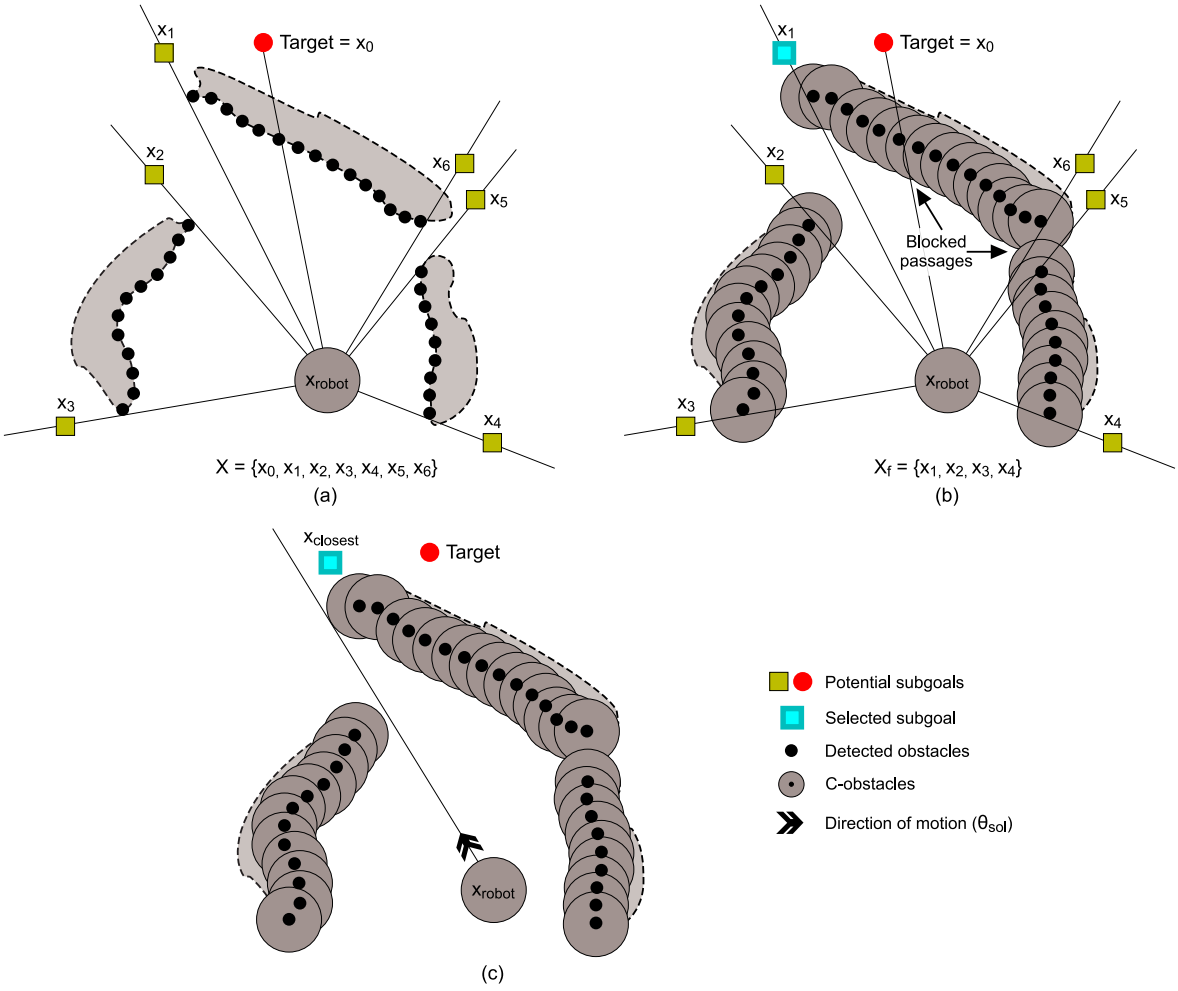


Figure 2.13: The two procedural steps of *ORM*: (a,b) subgoal selection and (c) motion computation.

2.5.3 The Smooth Nearness-Diagram Navigation Method

Author(s): Francesco BULLO [51] and Joseph DURHAM [51]

Reference(s): [51] 2008

Description: As can be guessed by the name, the *Smooth Nearness-Diagram method (SND)* constitutes a direct evolution of the obstacle-avoidance technique briefly referred to as *ND* which was outlined in section 2.5.1. As compared to the *ND* navigation scheme, *SND* essentially stands out by proposing a single motion law that is general enough to be applied to all possible configurations of surrounding obstacles[7], or equivalently, using terminology of *ND*, to all possible *situations* (remember that *ND* suggests dealing with

[7] The *Smooth Nearness-Diagram method* takes into account all nearby obstacles, not just two as done by *ND*. Consequently, this makes *SND* a safer and more robust strategy for navigation in tight spaces

separate motion laws for different scenarios). This change, away from being a mere unification of rules for obstacle-avoidance, has as a main advantage the removal of abrupt transitions in behavior when the robot navigates near obstacles, which necessarily leads to smoother trajectories.

Before concluding, it is worth to also cite the recently published work appearing in [52] (the *Closest Gap*, *CG*), which improves the safety of the paths generated by *SND* by considering the ratio of obstacles —threats— on the two sides of the robot, as well as by providing stricter deviation against the closest obstacles.

2.6 Sensory-based Motion Planning with Global Proofs

2.6.1 Bug1

Author(s): Vladimir LUMELSKY [53, 54] and Alexander STEPANOV [53]

Reference(s): [53] 1987, [54] 1991

Description: Essentially, the algorithm *Bug1* formalises the common sense idea of moving towards the goal unless an obstacle is found, in which case the vehicle circumnavigates the obstacle until the motion to the goal is once again allowable. The robot is assumed to be a point equipped with a perfect positioning system and contact sensors that can detect an obstacle boundary if the robot "touches" it.

The starting and the target points of the mission are labeled as S and T, respectively. On the other hand, let $L_0 = S$ and the *Main Line* (*m-line*) be the line segment connecting L_i to T. Initially, $i = 0$. The Bug1 strategy exhibits two different behaviors: motion-to-goal and boundary-following. The former moves the robot along the m-line towards T until either the goal or an obstacle is encountered. Assuming this last situation, let H_1 be the point where the vehicle first finds the obstacle and let us call this point a *hit point*. Under these circumstances, the boundary-following behavior is adopted by circumnavigating the obstacle until the robot returns to H_1. Next, the closest point to T on the perimeter of the obstacle is determined. This point is named *leave point* and is denoted as L_1. Finally, the vehicle traverses to L_1 and, from that location, it heads straight towards the goal again by reinvoking the motion-to-goal behavior. The previous procedure is repeated until the target is attained or the algorithm concludes that it is unreachable. Look at figure 2.14 and algorithm 2.3 for further details.

2.6.2 Bug2

Author(s): Vladimir LUMELSKY [53, 54] and Alexander STEPANOV [53]

Reference(s): [53] 1987, [54] 1991

Description: Like its Bug1 sibling, the algorithm *Bug2* also presents two behaviors: motion-to-goal and boundary-following. The former moves the robot towards the target point over the m-line which, contrary to Bug1, connects all the time S and T —notice that

Algorithm 2.3 Main steps for *Bug1*

0) Initialise $i = 1$ and $L_0 = S$.

1) From point L_{i-1}, move along the straight-line segment ST until one of the following occurs:

   a) T is reached. The algorithm stops.

   b) An obstacle is found. As a result, the hit point H_i is defined and step 2 is executed.

2) Follow the boundary of the obstacle to the left taking into account the next two possible situations which can arise:

   a) T is reached. The algorithm stops.

   b) The robot returns to H_i completing thus a loop around the obstacle contour. In this case, determine the point Q on the perimeter of the obstacle that has the shortest distance to T. Additionally, go to Q following the shortest way along the boundary. Finally, execute step 3.

3) If the line connecting Q and T intersects the current obstacle, then there is not a path to T, stopping, in consequence, the algorithm. Otherwise, several actions are taken: define the leave point $L_i = Q$, set $i = i + 1$ and, lastly, go to step 1.

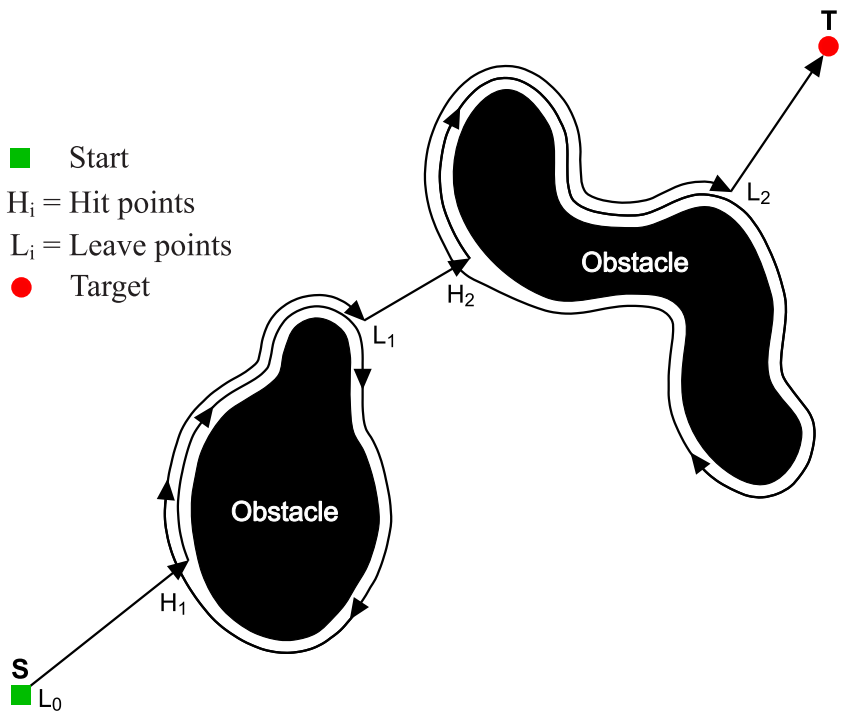


Figure 2.14: Robot's trajectory according to the algorithm *Bug1*.

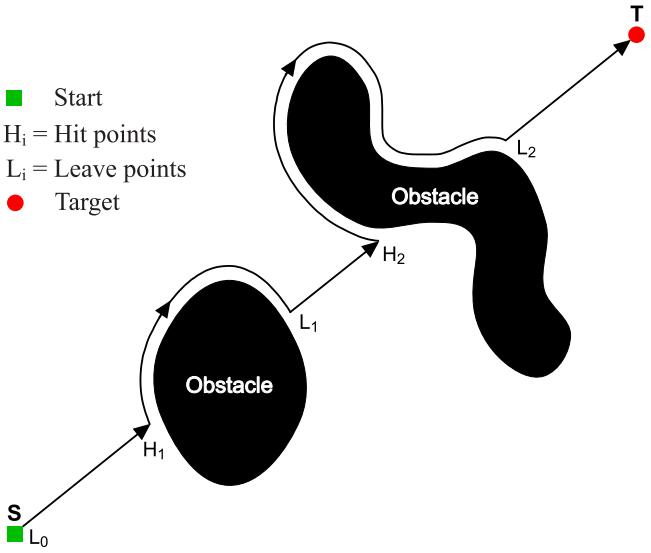


Figure 2.15: Path generated by the algorithm *Bug2*.

we use the same notation as in section 2.6.1. Regarding the boundary-following behavior, it is invoked when the robot finds an obstacle which impedes progress. In such a case, the vehicle goes around the sensed obstacle until reaching a point on the m-line closer to T than the initial point of contact with the obstacle —i.e. than the most recently defined hit point (H_i). At that moment, the robot proceeds to the goal again, repeating the circumnavigation process when another obstacle is detected on its way. See section 4.1 for a deeper description of the strategy Bug2 and figure 2.15 for an illustrative example.

2.6.3 VisBug

Author(s): Vladimir LUMELSKY [55] and Tim SKEWIS [55]

Reference(s): [55] 1990

Description: Going beyond the so-called "tactile" algorithms, mainly represented by Bug1 and Bug2 (refer to sections 2.6.1 and 2.6.2, respectively), the motion planner *VisBug* achieves improved navigation results by exploiting range data. The mobile robot is here assumed to be equipped with a vision sensor, which mimics a typical range finder in the sense that it provides the vehicle with the coordinates of those obstacle boundary points lying within a limited field of vision around the robot. In a few words, VisBug uses this —local— information to calculate shortcuts on a reference path generated by the algorithm Bug2. Specifically, the strategy first reconstructs in the current field of view of the robot the path which would be produced by Bug2. Later, VisBug moves the robot one step forward in the direction pointing to the farthest endpoint of such a Bug2 path. These two stages operate iteratively until either converging to the target (T) or realizing that T is definitely not reachable. By way of example, figures 2.16(a), (b), and (c) shows how the algorithm VisBug performs its first iteration in a scenario with just one obstacle. Additionally, figure 2.16(d) depicts the whole path that would be obtained.

2.6.4 DistBug

Author(s): Ishay KAMON [56] and Ehud RIVLIN [56]

Reference(s): [56] 1995

Description: The algorithm *DistBug* differs from the Bug1 and Bug2 approaches in, fundamentally, the two following aspects:

- The strategy assumes that the vehicle is equipped with finite range sensors instead of contact —zero range— sensors for the detection of obstacles.
- The behavior of the robot changes from boundary-following to motion-to-goal when range data guarantees that the next hit point will be closer to the goal than the previous one.

In this way, like Bug1 and Bug2, DistBug also ensures the mission completion whenever possible as well as the detection of those situations where the target is not reachable. Algorithm 2.4 gives the details.

Algorithm 2.4 Main steps for *DistBug*

{S and T represent, respectively, the starting and the target points of the mission}

0) Initialise $i = 1$ and $L_0 = S$.

1) Move along the straight-line segment $L_{i-1}T$ until one of the following occurs:

   a) T is reached. The algorithm stops.

   b) An obstacle is found. As a result, the hit point H_i is defined and step 2 is executed.

2) Follow the contour of the obstacle in the direction given by applying a minimum turn criterion. At the same time, take into account the next three possible situations which can arise:

   a) T is reached. The algorithm stops.

   b) The robot returns to H_i completing thus a loop around the obstacle boundary. In this case, the target cannot be achieved —it is unreachable— so that the algorithm is also stopped.

   c) By means of the robot's range sensors, a free-obstacle path in direction to T is detected guaranteeing that the next hit point (H_{i+1}) will be closer to the target than H_i. Under these circumstances, several actions are taken: define the leave point L_i in coincidence with the current robot position, set $i = i + 1$ and, lastly, go to step 1.

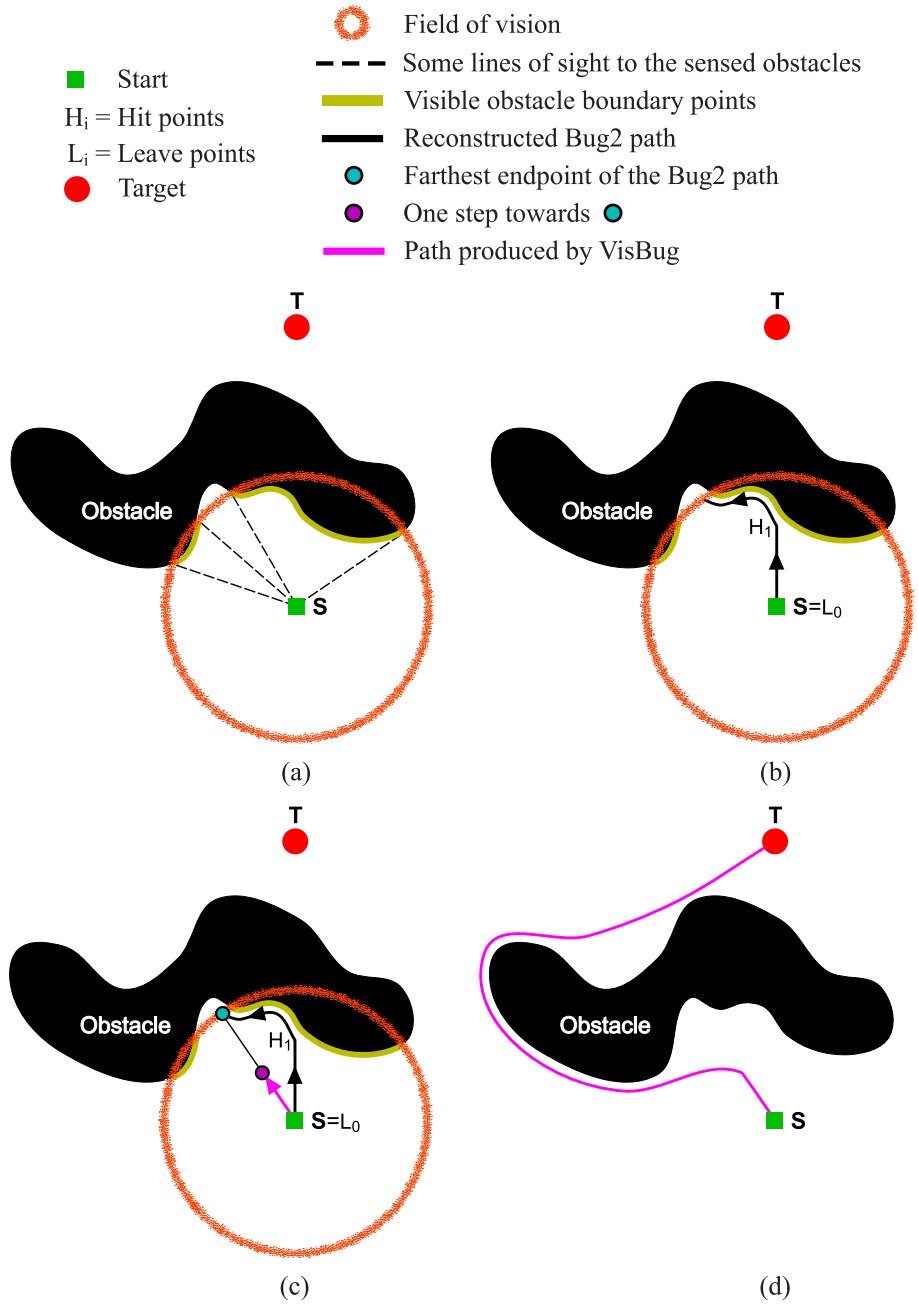


Figure 2.16: *VisBug* as an iterative three-stage algorithm: (a) sensing of the surrounding obstacles; (b) reference path generated by the strategy Bug2; (c) taking a shortcut on (b); (d) resultant trajectory.

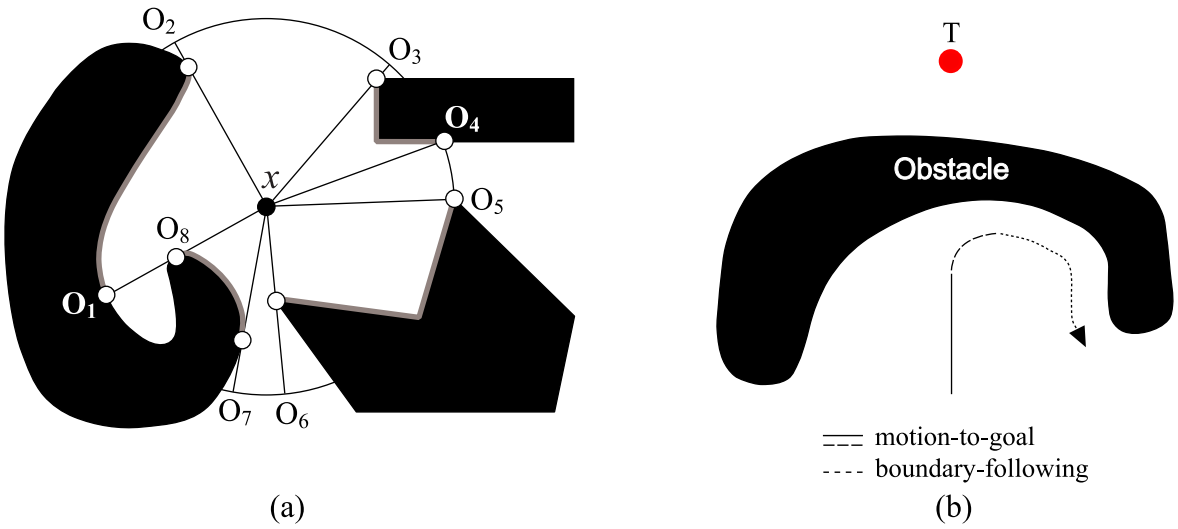


Figure 2.17: The *TangentBug* approach: (a) points of discontinuity of $\rho_R(x, \theta)$; (b) basic behaviors.

2.6.5 TangentBug

Author(s): Ishay KAMON [57], Elon RIMON [57] and Ehud RIVLIN [57]

Reference(s): [57] 1998

Description: *TangentBug* constitutes an improvement of the Bug2 algorithm (see section 2.6.2) by generating shorter paths to the goal using a range sensor with a 360 degree infinite orientation resolution. This range sensor is modelled with the so-called saturated raw distance function $\rho_R : \mathbb{R}^2 \times S^1 \rightarrow \mathbb{R}$. Considering a mobile robot located at $x \in \mathbb{R}^2$ with rays radially emanating from it, for each $\theta \in S^1$, the value $\rho_R(x, \theta)$ represents the distance to the closest obstacle along the ray from x at an angle θ. No obstacles are detected beyond a distance R according to the limited range of this kind of sensors.

The strategy TangentBug assumes that the robot can detect discontinuities in ρ_R. Figure 2.17(a) shows an example where the points O_i correspond to such losses of continuity, either as a result of one obstacle blocking another or the sensor reaching its range limit. On the other hand, notice that the sets of points on the boundary of the free space between O_1 and O_2, O_3 and O_4, O_5 and O_6, and O_7 and O_8, highlighted with gray thick solid curves, are the intervals of continuity.

Just like the other Bug approaches, TangentBug iterates between two behaviors: motion-to-goal and boundary-following. Initially, the former guides the vehicle. Consequently, the robot progresses along a straight line towards the target until it senses an obstacle in the goal direction. This means that the line segment connecting the robot and the goal intersects an interval of continuity $[O_i,\ O_{i+1}]$. Next, the vehicle moves to the point $O_j,\ j \in \{i, i+1\}$ that maximally decreases the heuristic distance function $d(x, O_j) + d(O_j, T)$, where d is another function that computes the Euclidean distance between two points and T denotes the target of the mission. The interval of continuity is

continuously updated as the robot moves to a particular O_j. The vehicle undergoes the motion-to-goal behavior until it can no longer decrease the heuristic distance to the goal or, in other words, until it finds a point that is like a local minimum of $d(\cdot, O_j)+d(O_j, T)$ restricted to the path that motion-to-goal dictates.

When the robot switches to boundary-following, it moves in the same direction as if it was in the motion-to-goal behavior (see figure 2.17(b)). While performing this motion, the algorithm also updates two values: $d_{followed}$ and d_{reach}. The former is the minimal distance from the target along the followed obstacle's boundary. The value d_{reach} is, on the other hand, the minimal distance from the target within the line of sight of the robot. When $d_{reach} < d_{followed}$, the vehicle finishes the boundary-following behavior.

To conclude, algorithm 2.5 completes the description of the TangentBug strategy.

Algorithm 2.5 Main steps for *TangentBug*

{Let C_T be the point where a circle centered at x of radius R intersects the straight segment connecting x and T. Notice that this is the point on the periphery of the sensing range that is closest to the target when the robot is placed at x. On the other hand, the endpoints of the interval of continuity in $\rho_R(x, \theta)$ located in the goal direction, if any, are denoted as O_i and O_{i+1}}

while T is not reached **and** T is not known to be unreachable **do**

{Motion-to-goal behavior}
repeat
{Notice that an infinite cost is associated to the function d when an obstacle is known to intersect the line segment defined by its two parameters}
Move towards the point $n \in \{C_T, \{O_i, O_{i+1}\}\}$ which minimises $d(x, n) + d(n, T)$.
until T is reached **or**
the direction that minimises $d(x, n) + d(n, T)$ begins to increase $d(x, T)$

{Boundary-following behavior}
Define the contour following direction in line with the most recent motion-to-goal direction.
repeat
Update $d_{followed}$, d_{reach} and $\{O_i, O_{i+1}\}$.
Move towards the point $n \in \{O_i, O_{i+1}\}$ that is in the chosen boundary direction.
until T is reached **or**
$d_{reach} < d_{followed}$ **or**
the robot completes a cycle around the obstacle, meaning that T cannot be achieved

end while

2.6.6 CBug

Author(s): Yoav GABRIELY [58] and Elon RIMON [58]

Reference(s): [58] 2005

Description: This algorithm called *CBug* is an interesting extension of the Bug1 approach presented in section 2.6.1. More precisely, given a start S and a target T, the strategy defines a virtual ellipse with focal points S and T, and area A_0 where the robot should afterwards look for the goal. The search is specifically carried out according to Bug1 (see algorithm 2.3), considering the boundary of the ellipse as an obstacle. As a result, if the target is achieved, the algorithm stops. Otherwise, the robot repeats the process on larger ellipses with areas $2^i A_0$ for $i = 1, 2, \ldots$ until T is finally reached or determined to be inaccessible from S. The main feature of CBug is that the resultant path length is at most quadratic with regard to the optimal trajectory, bounding thus the maximum difference between both solutions. To conclude, algorithm 2.6 describes the strategy in depth.

Algorithm 2.6 Main steps for *CBug*

0) Initialise $i = 1$, $S_i = S$ and $A(i) = A_0$, where A_0 is the initial area of an ellipse with focal points S and T. This ellipse is considered by the algorithm as a virtual obstacle.

Repeat

1) Search for T according to algorithm 2.3 (*Bug1*) and starting from S_i.
2) **If** T is reached, the process stops.
3) **Else** {T is determined not to be reachable}
   3.1) **If** the ellipse is not part of the current obstacle boundary
   T is definitely unreachable. In consequence, the process stops.
   3.2) **Else**
   Define S_{i+1} as the point where the algorithm *Bug1* finished
4) Set $i = i + 1$ and $A(i) = 2A(i - 1)$.
5) Compute the new ellipse.

End Repeat

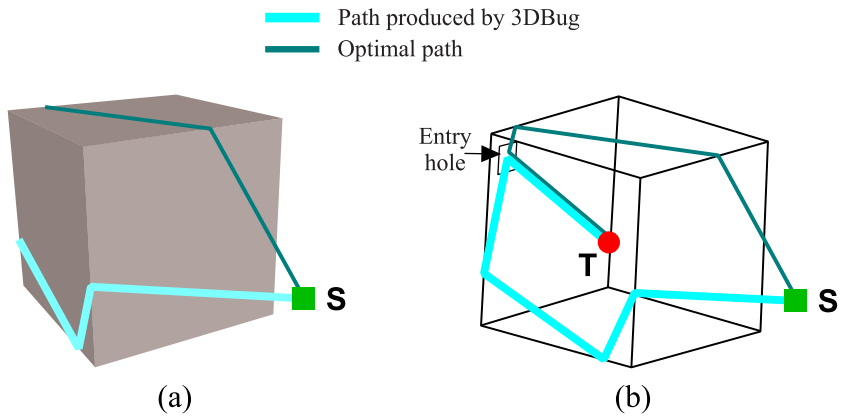


Figure 2.18: Solving a three-dimensional navigation problem with *3DBug* (a) solid view; (b) wired view.

2.6.7 3DBug

Author(s): Ishay KAMON [59, 60], Elon RIMON [59, 60] and Ehud RIVLIN [59, 60]

Reference(s): [59] 1999, [60] 2001

Description: The algorithm *3DBug* studies the problem of Bug-type three-dimensional navigation. Concisely, *3DBug* is able to effectively move a point-shaped robot equipped with a sensor with infinite detection range in a three-dimensional unknown environment populated by stationary polyhedral obstacles. Moreover, this is achieved while ensuring global convergence to the target. The algorithm uses two modes of operation, generically named *motion-to-goal* and *obstacle-surface-traversal*. During the former, the robot follows an estimation of the locally shortest path to the target[8]. Alternatively, after finding a blocking obstacle, the mode of operation of obstacle-surface-traversal is adopted. If so, the robot looks for a suitably exit point on the obstacle surface from where it can resume its motion towards the target. Figures 2.18(a) and (b) provide different views of the path generated by the algorithm *3DBug* in a scenario consisting of a closed box with a small hole near one of its corners —the target (T) is within the box, as opposed to the starting point (S) whose location does not allow the robot to see the box entrance. As can be observed, for comparison purposes, the globally shortest path is also represented in the figures. In quantitative terms, the solution given by *3DBug* is 1.67 times longer than the optimal one.

[8] The *locally shortest path* means the shortest free-obstacle path, from the current location of the robot to the target, that can be computed by only considering the currently visible obstacles

Traversability and *Tenacity*: Two New Concepts for Improving Navigation of Purely Reactive Control Systems under Limited Sensing Capabilities

The ability of a purely sensor-based / reactive control system for successfully solving a given navigation task depends on several factors, the most obvious being the technique adopted for avoiding obstacles. However, it is important to note that the particular features of the sensors mounted on the robot are also a critical factor influencing navigation. As an example of such an influence, changes are expected in the length and the smoothness of the resultant path, the risk of collision, and the likelihood of reaching the target point, when applying the same obstacle-avoidance technique on robots which differ in both the range and the resolution of the obstacle-detection sensors. As it seems clear, the less anticipation —meaning a shorter maximum detection range— and the less precise the robot's sensors estimate the position of the surrounding obstacles, the more significant / detrimental the changes will be in the three aspects mentioned above, typically used as descriptors of the quality of the navigation.

This chapter puts forward a novel framework for conducting complex maze-like navigation tasks, while wholly relying on local information —as it corresponds to a purely reactive control system— coming from low-cost proximity sensors (for instance, sonar / ultrasonic and infrared / IR sensors). Generally speaking, these sensors, beyond their inexpensive nature, are characterized by providing short and poor range measurements in non-controlled scenarios, mainly when compared to other usual sensors in mobile robotics such as laser range-finders. The key reason for facing the reactive navigation problem in an imprecise data context is to be able to devise robust strategies. As one can intuitively imagine, any strategy capable of efficiently and safely moving a robot equipped with low-cost sensors to the intended target, should similarly —or even better— behave when having wider and more accurate information about the environment.

In the following, as pointed out before, a generic framework is developed to help purely reactive control systems with the achievement of more difficult navigation tasks, while perceiving the environment —obstacles— through low-cost sensors. The proposal is based on two new concepts, named *Traversability* and *Tenacity* $\left(T^2\right)$. These concepts derive from the identification of the underlying causes that make the robot fall into a local minimum —in short, this represents a location, not corresponding with the one of the global target, where the robot gets stuck indefinitely (figure 2.4 illustrates this well-known problem, commonly referred to as

local minima). As repeatedly claimed in the related literature, local trapping situations are the most limiting factor for successful navigation under the purely reactive control paradigm. In this sense, the concepts T^2 arise with the aim of being an effective method for eliminating such undesirable situations, while keeping the exclusively local character which distinguishes the indicated paradigm from all others. So, with no doubt, the suggested T^2-based framework is going to noticeably increase the complexity of the navigation tasks that a purely reactive control system is expected to perform.

Before concluding this overview of the content of chapter 3, let us highlight the inherent generality of the concepts T^2. To this respect, these concepts can be plainly incorporated into many of the currently-existing purely reactive approaches. As an evident proof of such generality, this chapter additionally describes the way in which two popular, and quite different, navigation techniques can become non-susceptible to the local minima problem on the basis of T^2. More precisely, the enhanced techniques are the classic *Potential Fields Method (PFM)* and the *Dynamic Window Approach (DWA)* (see sections 2.1 and 2.4.1 for a brief introduction to both strategies). Observe that, despite *PFM* and *DWA* rely on different principles[1], the same benefits —meaning the overcoming of all local trapping situations— result from embedding T^2 within these two strategies.

The rest of the chapter is organized as follows: section 3.1 looks at the concepts of *Traversability* and *Tenacity* $\left(T^2\right)$, while presenting them as a part of the so-called *navigation filter*; afterwards, in section 3.2 / 3.3, the navigation filter is used to construct the local-minima free version of *PFM* / *DWA*; and, finally, section 3.4 discusses the points in favor and against T^2 as a new purely reactive framework for robot navigation.

3.1 The Navigation Filter

The inability to move the robot away from the target direction in a non-momentary and strategic manner is the main reason causing local trapping situations in the purely reactive control paradigm. By way of example, taking the *Dynamic Window Approach* as a representative case of a purely reactive navigation method, figure 3.1 evidences how a robot adopting *DWA* does fail in overcoming an L-shaped obstacle. As can be observed, the robot is not able to escape from the obstacle concavity by moving back, i.e. by moving in the direction opposite to the target. Along this section, a solution is given to this problem by means of the *navigation filter*. This new module is purposely designed to guide the robot out of any local minimum, irrespective of both the shape and the size of the obstacle /s creating the corresponding potential trapping situation.

3.1.1 About Inputs and Outputs

Let θ_{target} denote the direction from the current position of the robot to the target location, just as depicted in figure 3.2(a). Then, at each iteration of the robot control cycle, θ_{target} is provided as an input to the navigation filter. On the other hand, regarding the outputs, the navigation filter generates the result of transforming θ_{target} into a more promising direction of motion, referred to as θ_{sol}. Figure 3.2(b) summarizes the single-input / single-output scheme of the navigation filter.

[1] According to the widely-accepted classification given in [61], *PFM* is a directional method while *DWA* is a velocity-space method

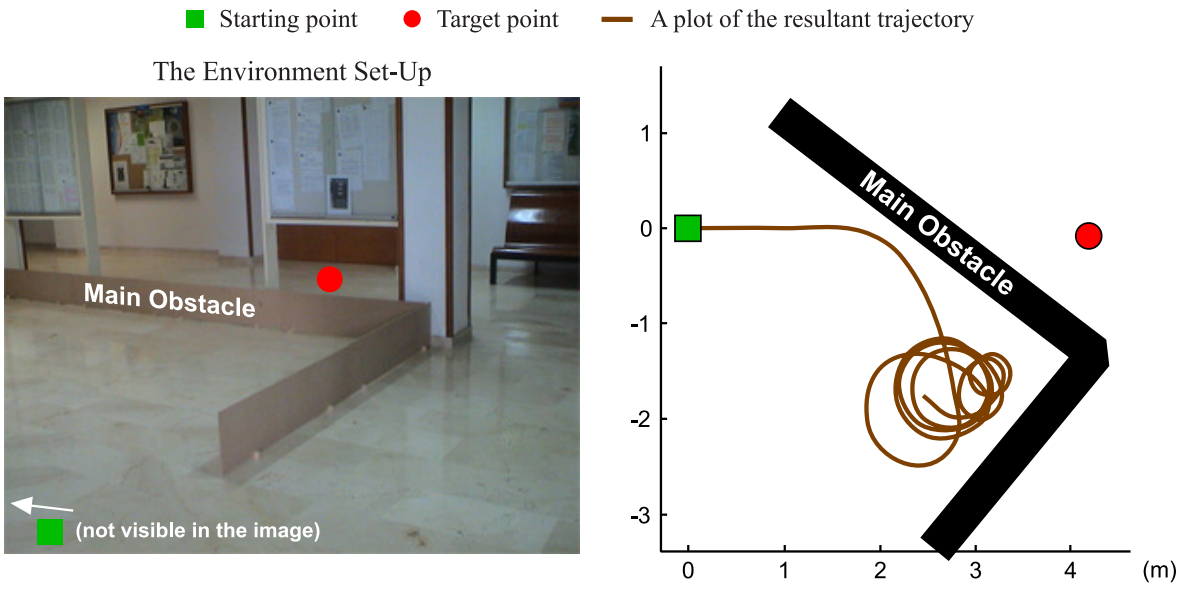


Figure 3.1: Identifying what is lacking in purely reactive control systems to avoid the robot to be trapped due to local minima (in the experiment above, *DWA* is tested in a troublesome scenario using a *Pioneer 3-DX* robot).

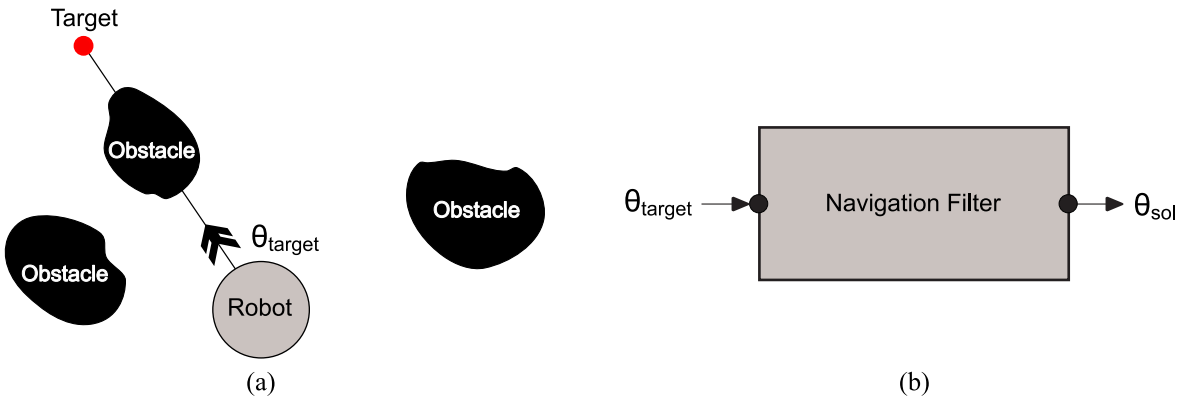


Figure 3.2: The navigation filter at first glance: (a) meaning of θ_{target}; (b) θ_{target} and θ_{sol} as input and output, respectively.

3.1.2 Basic Principles

The fundamental task of the navigation filter is the appropriate alteration of its input θ_{target} —this is assumed to be the default / the most desired direction of motion for the robot. Such change on θ_{target} is carried out only when becoming aware of the detection of obstacles in that precise direction. If so, the output θ_{sol} is computed in accordance with the principles of *Traversability* and *Tenacity*, jointly called T^2 (see footnote [2]). Concisely, the first one suggests banning those directions where an obstacle has been found as well as choosing some obstacle-free directions near the desired one —i.e. θ_{target}— when it has been banned. In this last case, the *tenacity* principle ultimately determines what obstacle-free direction will be selected among all the available alternatives. Next, both principles are described in depth.

3.1.2.1 *Traversability*

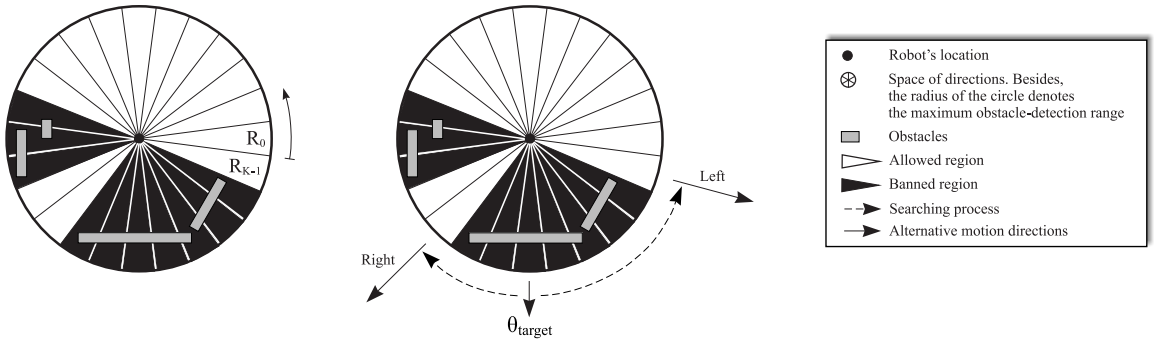
The application of the *traversability* principle requires the division of the space of directions around the robot into K identical angular regions as it is illustrated in figure 3.3(a). Each region comprises a disjoint range of directions which can be classified as *allowed* or *banned*. Specifically, a region is said to be *allowed* when all the directions in its range are obstacle-free. On the contrary, when this condition is not satisfied, the region is alternatively classified as *banned*.

Based on the previous information, this principle is intended to forbid the robot's movement in directions where the presence of obstacles has been recently determined, avoiding thus unnecessary and unsuccessful displacements in the task of looking for a feasible path to the target point. With this purpose in mind, the viability of the input θ_{target} is studied according to the above-mentioned premise. Changes are required only if such direction of motion lies in a *banned* region (otherwise, the navigation filter does not act, producing as output the same input direction —$\theta_{sol} = \theta_{target}$). More exactly, in the non-trivial case where θ_{target} results to be *banned*, two alternative motion directions, generically labeled as *left* and *right*, are obtained as the outcome of a double searching process, clockwise and counterclockwise, for the first *allowed* region starting from θ_{target} (see figure 3.3(b) for a clarifying example). The final decision about choosing one direction or another as θ_{sol} absolutely depends on the principle of *tenacity*, which will be explained next.

3.1.2.2 *Tenacity*

Taking figure 3.1 as an example, the trajectory of a robot that has been trapped can be seen, in broad terms, as the result of an endless sequence of forward and backward movements. These movements alternate each time the vehicle moves either excessively away from the target direction (θ_{target}) or very close to the obstacles blocking θ_{target}. The *tenacity* principle precisely tries to give a solution to that oscillating and hesitant behavior by preventing the robot from abruptly changing its direction of motion. Besides, in this manner, progress is always ensured, removing thus the major cause of local trapping situations. As for the implementation details, remember that two alternative motion directions labeled as left and right

[2] Notice that concepts roughly related with the principles of *traversability* and *tenacity* can be separately found in several studies in the field of obstacle avoidance such as, for instance, in [16] and [48]. These concepts, nevertheless, have never been combined —to the author's best knowledge. The proper definition and combination of the principles T^2 makes the robot behave in such a way that it gets never trapped into a local minimum. Conversely, [16, 48] suffer from the local minima problem



(a) (b)

Figure 3.3: Understanding the *traversability* principle: (a) division of the space of directions into K regions $(R_0, \ldots, R_{K-1})$, which are labeled as *allowed* or *banned*; (b) selection of two obstacle-free motion directions, named *left* and *right* $\Big($in a few words, each of these directions corresponds to the midpoint of the range of the counterclockwise (left) / clockwise (right) *allowed* region that is closest —angularly speaking— to $\theta_{target}\Big)$.

are found when θ_{target} lies in a *banned* region. Under these circumstances, one of such directions has to be selected as output of the navigation filter —θ_{sol}. To this end, the principle of *tenacity* is applied, which consists in choosing left or right in coincidence with the last decision made. Despite the obvious simplicity of the concept, it has proven fully effective.

3.1.2.3 Temporarily Remembering the Obstacles

As was mentioned in section 1.3, purely reactive approaches react directly to the world as it is sensed, avoiding the need for intervening any kind of abstract representational knowledge. The sentence "what you currently see is what you get" faithfully sums up this idea. Therefore, in this context, only the use of the local information provided, at the current time instant, by the robot's sensory equipment is permitted for solving the intended navigation task. On a practical implementation, however, this constraint is, in general, slightly relaxed by allowing that these methods can make their decisions based upon a small memory / buffer containing all the environmental information collected by the robot's sensors during a configurable time window —an example of a purely reactive control system, namely the *Nearness Diagram method (ND)* (refer to section 2.5.1), with such an implementation pattern can be found in the widely-employed *Player / Stage* software. In this way, the aforementioned sentence describing the inherent meaning of purely reactive navigation can be rewritten, from an implementation point of view, as "what you have recently seen is what you get".

Adopting this practical perspective, our navigation filter considers information regarding the obstacles beyond the current robot's field of view, by memorizing the approximate location of those obstacles that have been freshly detected. This information is used, after a change in position of the robot, to recalculate the type —*banned* or *allowed*— of the regions that conform the whole space of directions. By way of example, figure 3.4 clearly illustrates the two following related facts: on the one hand, the capability of the navigation filter for remembering the existence of obstacles in directions where, presently, an obstacle-free space is perceived by the robot's sensors; on the other hand, the up-to-date type of the regions which consistently reflects all the recently-obtained obstacle information.

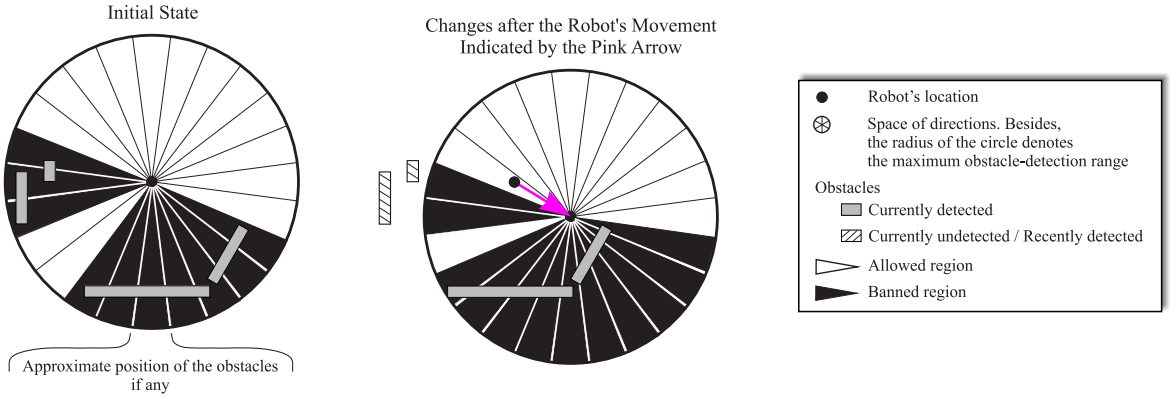


Figure 3.4: Short-term buffering of the location of the obstacles which are sensed as navigating.

As expected from a local framework, the navigation filter holds (obstacle) data, but just temporarily. To this respect, nevertheless, it is important to note that such a temporality is not achieved by the simple definition of a time window of fixed length, where each data point remains in memory during a fixed time horizon —being later removed—, as actually done, for instance, in *ND*. On the contrary, the navigation filter accumulates environmental data for some variable time —once this time is over, all data previously accumulated is discarded. To be more precise, this accumulation process is constrained to a time window whose length varies depending on the specific characteristics —basically, shape and size— of the obstacles that are impeding, at a given time, the progress of the robot towards the target. In short, the navigation filter does not get rid of any data until ensuring that the currently blocking obstacles have been overcome. Further details about this subject will be explained in section 3.1.3.

Finally, relating to the above, let us highlight that the adaptive temporary storage of data is a necessary feature for the navigation filter in order to cope with the pursued goal of escaping from all local minima —regardless of both the shape and the size of the obstacles causing these potentially trapping situations, as well as the maximum detection range of the robot's proximity sensors— while requiring, at the same time, an amount of memory continuously fitted to the recently-experienced environmental circumstances.

3.1.3 Analyzing the Induced Robot's Behavior

A robot moving in the direction given by the navigation filter, referred to as θ_{sol} —notice that this T^2-compliant direction is continuously updated to accommodate for both the new environmental data acquired by the robot's sensors, and the changing location of the robot—, exhibits the next three behavioral traits:

1. When the robot is navigating far from obstacles, it heads for the target point following a straight-line path. During this period of time, the navigation filter remains, in some sense, inactive by generating as output the same input motion direction, which means that $\theta_{sol} = \theta_{target}$.

2. After the detection of an obstacle, the robot follows its contour in a certain direction. This contour following process emerges as a direct consequence of the changes performed

by the navigation filter on θ_{target} according to T^2. Let us now focus on the first time that θ_{sol} is going to differ from θ_{target}. In this particular case, once having obtained the *left* and *right traversability*-based choices, the principle of *tenacity* cannot be subsequently applied due to the lack of a previous decision about the convenience of selecting either the *left* or *right* choice as the output θ_{sol}. A set of different criteria to make this first / undefined decision will be discussed later, however, for now, it is important to know that such a decision does determine the specific direction taken by the robot to move along the boundary of the involved obstacle.

3. Lastly, the robot realizes that the obstacle has been suitably circumnavigated when the direction to the target becomes free of obstacles, or in other words, when θ_{target} lies in an *allowed*-type region. At that moment, the filter is reset losing thus all the previously kept information —i.e. the up-to-now perceived part of the obstacle—, which is no longer needed in the assumed context of local navigation.

These stages are sequentially executed in the order specified as many times as obstacles the robot finds on its way towards the target point (look at figure 3.5 for a simple example).

3.1.4 Some Relevant Considerations

Once the fundamentals of the navigation framework T^2 have been examined, more specific details about it are considered in the following:

- As clearly pointed out in section 3.1.3, after the detection of a new obstacle, the first computation of θ_{sol} cannot be properly made just in accordance with the principles T^2. This is essentially due to the fact that the principle of *tenacity*, because of relying on its own previous decisions, is not able to solve the problem of choosing, for the first time, between either the *left* or *right* alternative motion directions which derive from applying the *traversability* principle (see figure 3.5(b) where it is revealed the lack of answer of the *tenacity* principle at step 3). Therefore, it seems evident that an additional criterion is required to deal with such left-out situations. With this aim, let us, first of all, emphasize the repercussions that come from selecting one of the two —*left* and *right*— *traversability*-based choices in the given context of first-time computations of the output θ_{sol}. Specifically, each of these decisions ultimately determines the resultant following direction of the contour of the corresponding new obstacle that is impeding the robot's advance. Next, keeping this in mind, an analysis is presented with regard to the pros and cons of three common-sense criteria that could help the navigation filter decide the precise direction in which the robot would have to move along the boundary of the detected obstacles. Briefly stated, these criteria are as follows:

  The *minimum-turn* criterion. In this case, the contour following direction to be taken by the robot coincides with the one that involves the smallest / minimum turning angle. As most important advantages of this criterion, we can mention a slightly reduced energy consumption and a lower risk of collision with obstacles because of performing a simpler robot's maneuver. As for the drawbacks, the *minimum-turn* criterion unfortunately favors that some parts of the boundary of an obstacle are traversed more than once, as shown in figure 3.6(a). What is worse, such a repeated traversal may easily lead to cyclic behaviors, preventing thus the robot from converging to the target point (refer to figure 3.6(a) again).

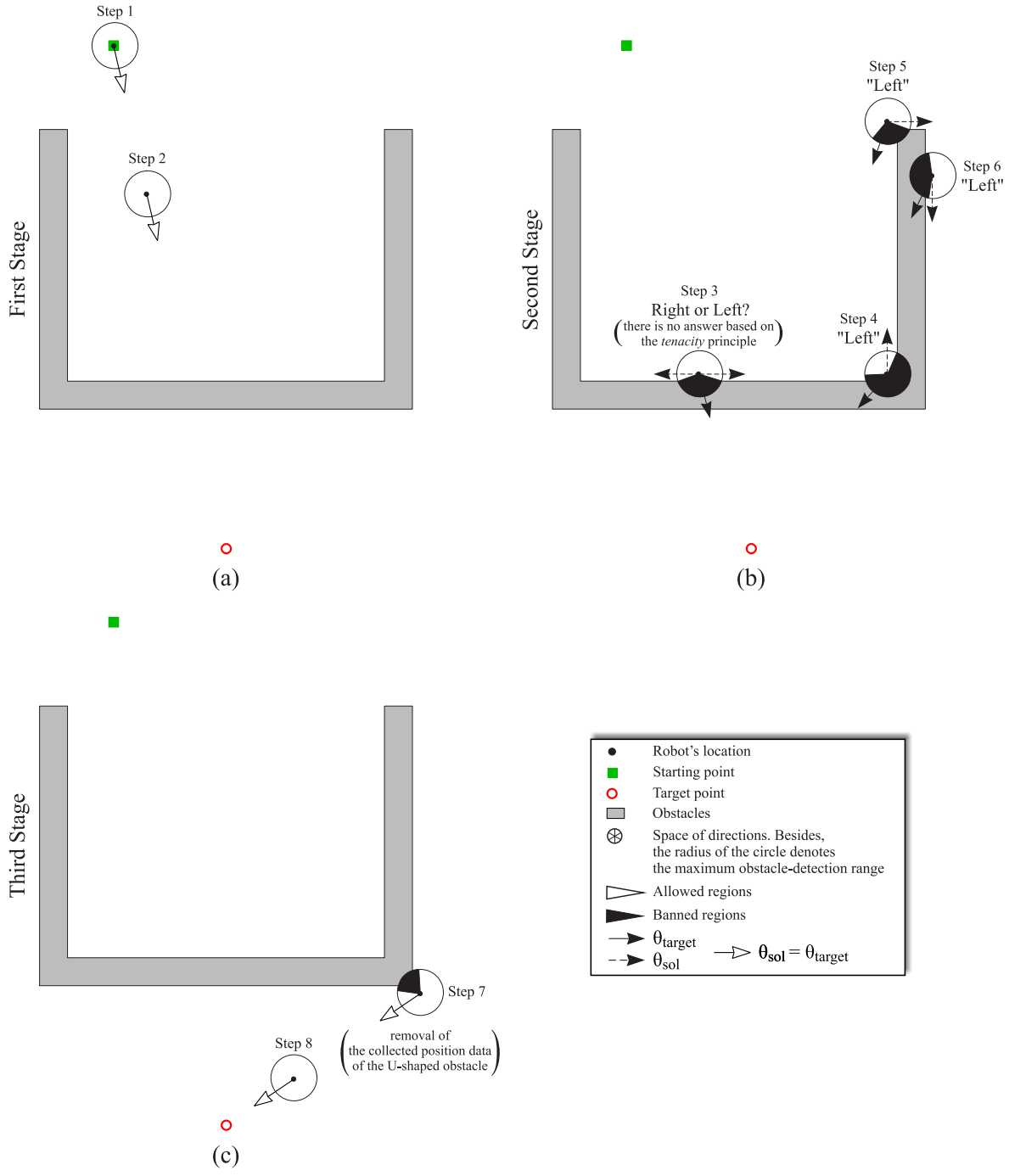


Figure 3.5: Pattern behavior of a T^2-based purely reactive navigation method while escaping from a large U-shaped obstacle: (a) direct path to the target; (b) following the contour of the obstacle to the robot's left (observe that this direction is defined in step 3 by supposedly choosing the option labeled as *left* between the two alternatives that result from the application of the *traversability* principle); (c) reset of the navigation filter returning, later, to (a) again where the target point is definitely reached.

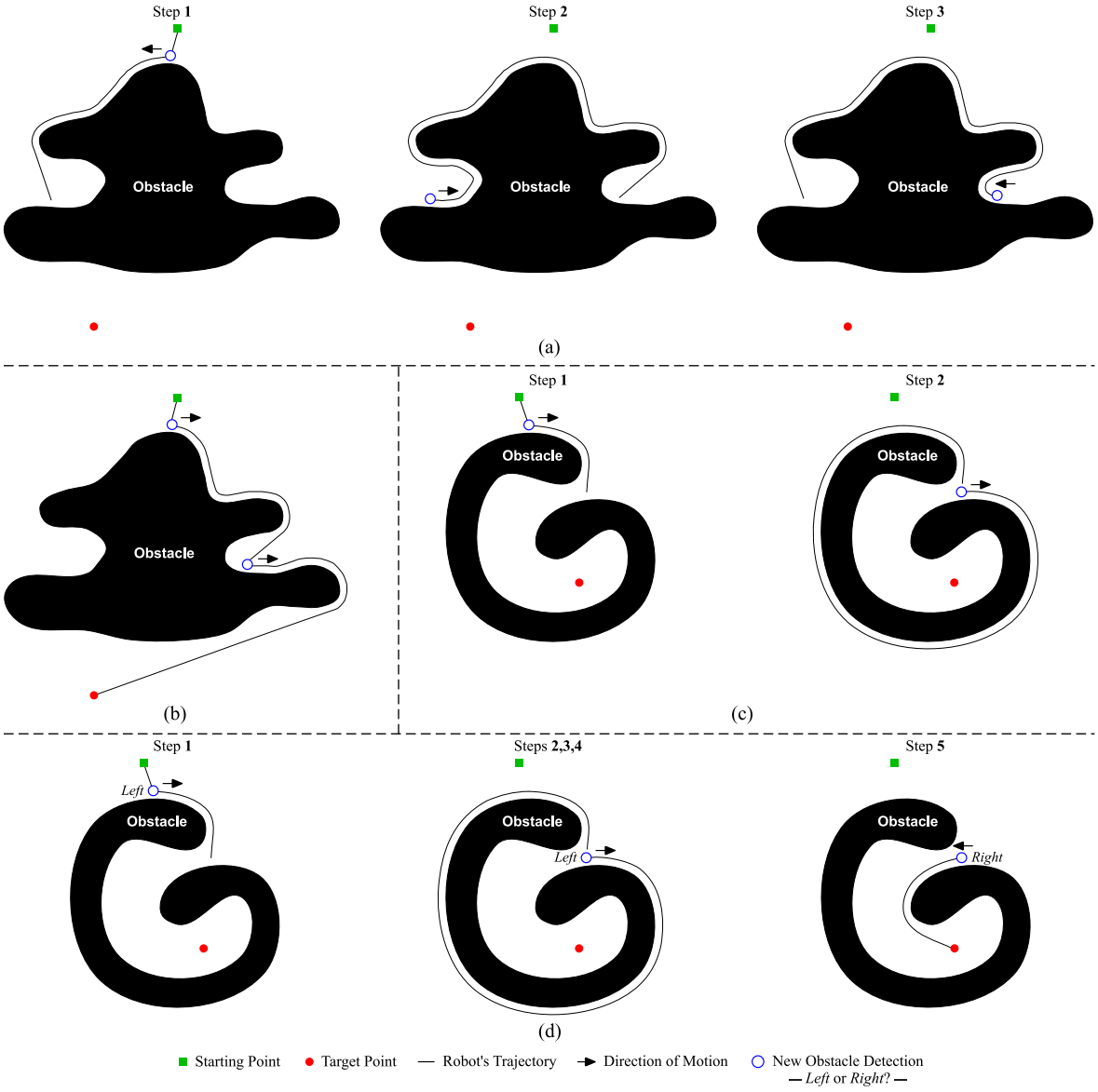


Figure 3.6: Exemplification of three alternative criteria to decide on the contour following direction: (a) *minimum turn*; (b) and (c) *fixed beforehand* —set to *left*, in both cases—; (d) *random*. As a general remark on the figure, it is important to stress that all the trajectories have been drawn by hand according to the known —as described by section 3.1.3— way of acting of a robot that adopts the principles T^2. Additionally, going deeply into some particular cases, notice that in (a) / (c), after step 3 / 2, the second step is executed again giving thus rise to a cyclic behavior.

The *fixed-beforehand* criterion. Based on this criterion, the direction to follow the boundary of the obstacles is always the same. Moreover, such a constant direction should be explicitly defined by means of a parameter prior to conducting the navigation task. As it seems obvious, this is one of the simplest ways of facing the problem of selecting one contour following direction or another —i.e. either *left* or *right*—, since the decision does not require any calculation, just take the value given to the aforesaid parameter (notice that, in computational terms, the formerly explained *minimum-turn* criterion is plainly more demanding than the *fixed-beforehand* criterion, due to the need for estimating the angle in which the robot approaches the obstacle surface). On the other hand, the second main benefit of the *fixed-beforehand* criterion is its remarkable tendency to elude cyclic behaviors. By way of example, figure 3.6(b) illustrates how a *fixed-beforehand*-type robot gets a global and continuous progress to the target, while avoiding to revisit all previously traversed obstacle boundary segments. To conclude, however, it is important to highlight that cyclic behaviors may actually appear under the *fixed-beforehand* criterion, but merely in less natural environments with very intricate obstacles such as the one of figure 3.6(c).

The *random* criterion. As can be guessed by the name, in this context, the specific direction taken by the robot to circumnavigate the contour of a new detected obstacle is randomly chosen. Among the most interesting features that come with this criterion, the two following stand out: first of all, simplicity; and, secondly, the so-called probabilistic resolution of cyclic behaviors, which essentially means that the probability of interrupting such repeated patterns converges to 1 as time goes to infinity. Regarding this last feature, figure 3.6(d) exhibits the ability of a robot based on the *random* criterion to end up getting to the target point after looping several —three— times around the boundary of the G-shaped obstacle. Finally, observe that the same figure also explains the major inconvenient in the use of the *random* criterion. As can be seen, quite lengthy trajectories may result.

Table 3.1 provides a summary of the key points in favor and against of the three criteria discussed earlier. As can be inferred from the table, *minimum-turn* is manifestly the worst criterion that has been considered, because of having a high computational cost —as compared to the other suggested criteria—, and not preventing the robot from frequently getting stuck in behavioral cycles. Both of these problems are significantly reduced under the *fixed-beforehand* criterion, and under the *random* criterion as well. Accordingly, now let us find out which of these two advantageous criteria should be finally adopted by the navigation filter. In this sense, by observing table 3.1, one can rapidly conclude that there is not a clear choice between the *fixed-beforehand* criterion and the *random* criterion on the basis of their distinguishing benefits / drawbacks. Such an impossibility to make a decision is essentially due to the fact that the preference in the use of one criterion or the other depends on the particularities of the environment in which the robot is going to navigate. To be more exact, when assuming a navigation environment just consisting of non-intricate obstacles[3], the *fixed-beforehand* criterion is generally better suited than the *random* criterion, since the former is expected to successfully guide the robot to the target point through a shorter path. In the rest of

[3] Here, the term *intricate* is employed to refer to an obstacle that encircles the target point, such as the one of figure 3.6(c) (see also figure 3.6(a) for an example of a non-intricate obstacle)

Table 3.1: A comparison of the *minimum-turn*, *fixed-beforehand*, and *random* criteria from the viewpoints of both their associated computational cost and their effectiveness in avoiding cyclic behaviors.

Criterion	Relative Computational Demands	Capacity for Avoiding Endless Cycles
minimum-turn	High	In general, bad
fixed-beforehand	Low	Good in most of the typical environments
random	Low	In general, good but at the cost of long paths

scenarios, however, the *random* criterion is a preferably choice, with the main aim of increasing the likelihood of converging to the target.

From above, it seems evident that, among the three proposed criteria for choosing the contour following direction, the *minimum-turn* criterion stands out for being the least promising solution. As for the two remaining criteria, namely *fixed-beforehand* and *random*, it is well-known that none of them is going to fully outperform the other on a heterogeneous set of scenarios. Concisely, such a heterogeneous set includes three different environmental settings: on the one hand, those exclusively involving either intricate or non-intricate obstacles; and, on the other hand, the special setting which mixes both types of obstacles. In a few words, the most outstanding conclusion that issues from comparatively analyzing the *fixed-beforehand* and *random* criteria is that they offer quite different trade-offs between completeness and path length performance. Bearing this in mind, in the forthcoming sections —particularly referring to sections 3.2 and 3.3 where some T^2-based experimentation is carried out—, the navigation filter will be tested by using both criteria. The reason for this double testing is to demonstrate that, irrespective of the criterion —or trade-off— definitely adopted, the navigation filter does achieve reasonable results in environments where conventional purely reactive methods distinctly fail.

- A special consideration requires the situation illustrated in figure 3.7(a). As can be seen, after step 5, the robot is not allowed to head for the target point because it is known there are obstacles in that direction, although they are really far away. Consequently, this mission will never be successfully completed. In order to properly solve this problem, the obstacle information maintained by the navigation filter is partially contrasted with the immediate / local reality of the environment in the next way: once having applied both the *traversability* and the *tenacity* principles, the *banned* region that is closest to θ_{sol} is checked for feasibility so that its state is changed to *allowed* if no obstacles are detected in its vicinity for the corresponding range of directions. Figure 3.7(b) depicts the path of the robot after incorporating this control logic for the mission initially taken as example.

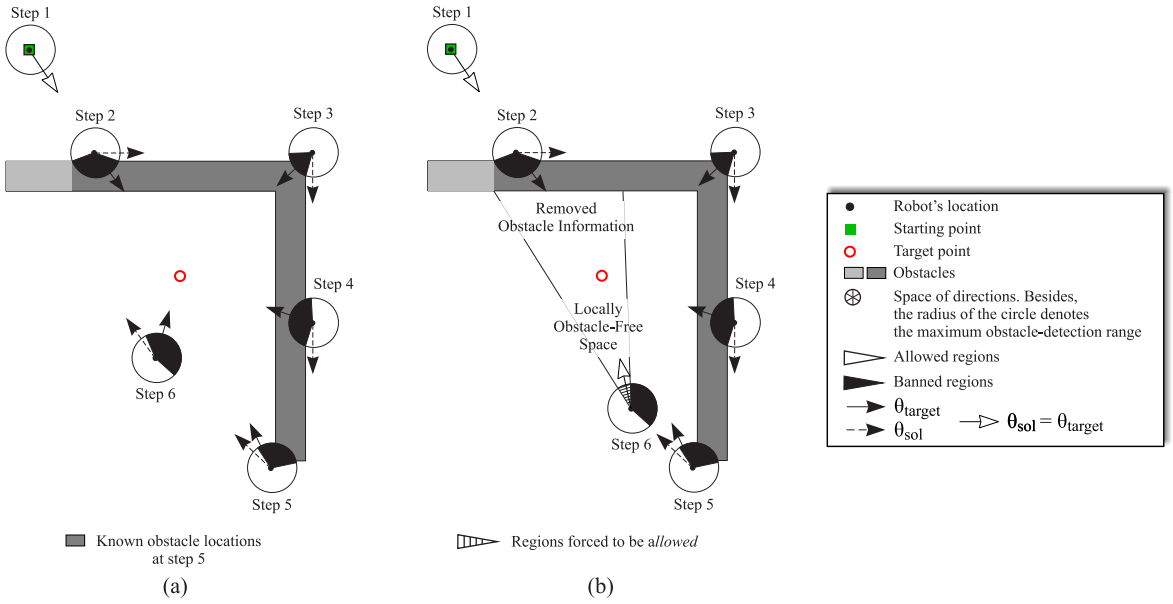
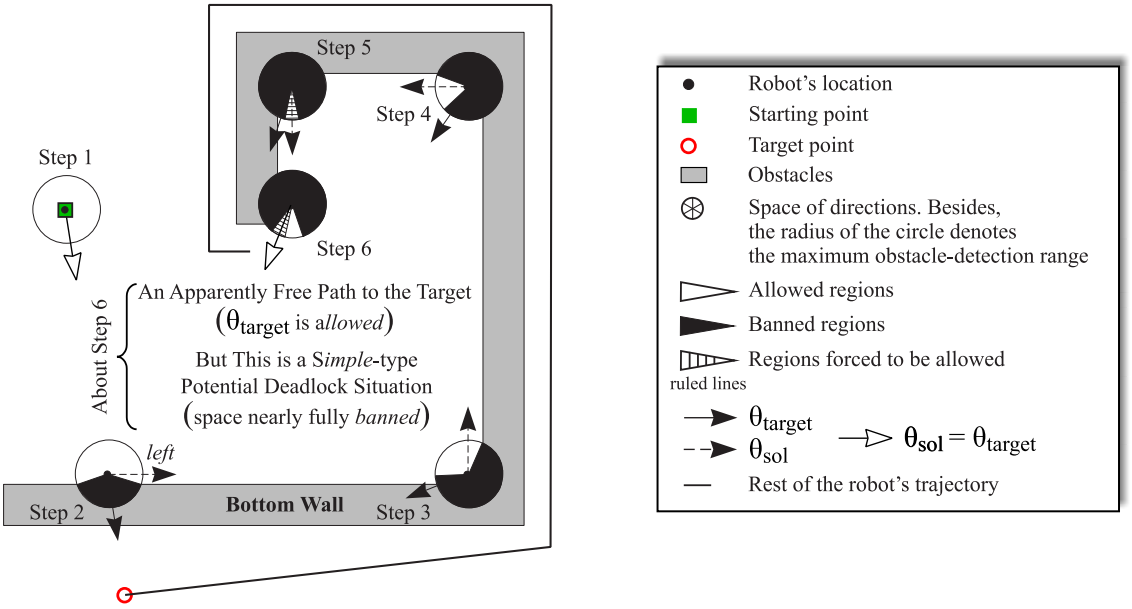


Figure 3.7: Removal of the obstacle information gathered by the navigation filter which unnecessarily restricts the motion of the robot.

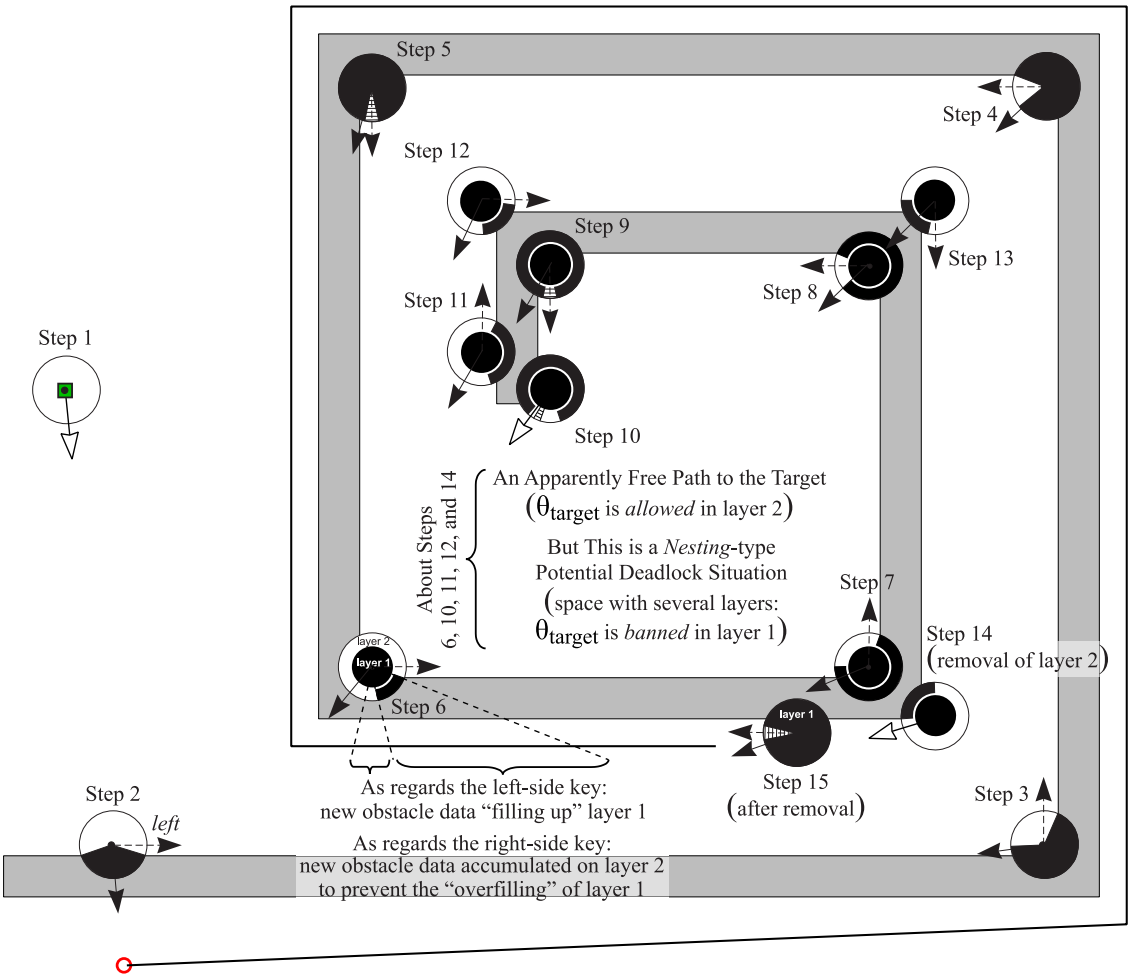
- The navigation filter is able to properly identify and act on the so-called *potential deadlock situations*. In brief, these situations are essentially due to obstacle concavities and may result in temporal, or even permanent, cyclic behaviors. To be more precise, a potential deadlock situation arises when the robot, while moving along the contour of an obstacle, makes a back loop which finishes facing the target point, as depicted in figure 3.8(a). By analyzing such a figure, it seems clear that the robot, at step 6, should decide between continuing following the boundary of the obstacle, or directly progressing towards the target. To this respect, keeping in mind the general aim of decreasing the chances for cyclic behaviors, there is no doubt that the former choice is more suitable / less risky than the latter one (observe that, under the second choice, steps 3 to 6 will be repeated again if the robot, after reaching / detecting the bottom wall of the obstacle on its straight-line way to the target, chooses *left* as the new contour following direction[4]). In closing, the preceding discussion definitively supports the fact that the navigation filter adopts the conservative strategy of not abandoning the contour following process in the face of potential deadlock situations (accordingly, figure 3.8(a) provides a multiple-line-segment representation of the trajectory that would be performed by the navigation filter after step 6).

  In the next lines, further details are given about the range of different potential deadlock situations that can be feasibly managed by the navigation filter. In this regard, let us redefine what the navigation filter actually considers as a potential deadlock situation —in essence, these situations are characterized by significantly favoring the generation of behavioral cycles. Before redefining the concept, however, remember that a potential deadlock situation was first stated to be caused by obstacles having a simple

[4] Either the fixed-beforehand criterion or the random criterion will determine the precise direction —*left* / *right*— taken to follow the wall



(a)



(b)

Figure 3.8: Distinguishing (a) *simple* and (b) *nesting* potential deadlock situations (notice that in both exemplifying cases, the navigation filter is conveniently assumed to choose the *left* contour following direction at step 2).

back-loop form, as illustrated in figure 3.8(a). To be fair, the navigation filter goes far beyond this limited perspective by generally understanding a potential deadlock situation as being produced by obstacles that are looping back according to a spiral pattern. So, based on this broader perspective, the obstacle of figure 3.8(b) certainly constitutes a potential deadlock situation because of its double-loop spiral-like shape (additionally, notice that the obstacle of figure 3.8(a) keeps still labeled as potentially deadlocking, since it is a degenerate / one-loop case of a spiral). In short —and as will be seen later—, the navigation filter prevents the robot from getting stuck in all spiral-inspired potential deadlock situations.

Now, once having clarified the meaning of the concept of potential deadlock situation, the (so-far omitted) issue of how the navigation filter recognizes these troublesome situations is carefully addressed. Nevertheless, prior to focussing on this subject, let us classify all potential deadlock situations into two groups. Specifically, this classification is made on the basis of the spiral-like form that is supposed for any deadlock-type obstacle. Furthermore, the number of loops in the spiral is the particular feature used for classification purposes. In view of that, the two following groups are defined: on the one hand, the so-called *simple* group, which includes those potential deadlock situations / obstacles resembling a spiral with exclusively one loop; on the other hand, the group referred to as *nesting*, which comprises every multi-loop spiral-shaped obstacle. At this point, it is important to highlight that the navigation filter employs different methods for identifying *simple* and *nesting* potential deadlock situations. Next, these methods are separately studied:

Concerning *simple* spiral-shaped obstacles. In a few words, the navigation filter knows that the robot is currently circumnavigating the contour of an obstacle with *simple* spiral-like shape when the space of directions is almost fully *banned* (take figure 3.8(a) as an example).

Concerning *nesting* spiral-shaped obstacles. First of all, let us discuss an additional characteristic of the navigation filter named *multi-layer*. In practical terms, this feature is exhibited only when having a space of directions entirely *banned*. Specifically, in such circumstances, the navigation filter exploits the concept of *layer* to provide a conceptual separation between the obstacle information that was previously accumulated and the one that is going to be gathered in the future (notice that, in essence, a layer corresponds to a representation of a subset of the available obstacle information in the form of a 360-degree / polar-like space, which ultimately designates both feasible / *allowed* and unfeasible / *banned* directions of motion for the robot). By way of example, figure 3.8(b) shows clear evidence of the multi-layer feature in a rather difficult scenario with a multi-loop obstacle. As can be seen, the space of directions becomes fully *banned* at step 6 —i.e. after completely traversing the outer loop of the spiral-shaped obstacle. As a direct consequence of this fact, the navigation filter acts as follows: on the one hand, the current space of directions —obviously, this includes all data causing its fully-*banned* state— turns into layer 1; on the other hand, a further layer is defined, namely layer 2, with the definitive intention of being used for representing / allowing for the motion restrictions imposed by the obstacle data to be collected from step 6 onwards (to this respect, observe, by looking at figure 3.8(b) again, that layer 2 is properly updated from step 6 to 14 in accordance with the newly as

well as the early gained obstacle information[5] —remember that data removal also occurs during navigation, as indicated by the ruled lines that, from time to time, appear in the space of directions).

Leaving momentarily aside the example of figure 3.8(b), let us now generally describe the two most relevant functional aspects —particularly referring to the creation and destruction of layers— of the multi-layer feature. To begin with, assume that X denotes the number of currently existing layers —$X \geq 1$. Then, the navigation filter will create layer $X+1$ when becoming aware of a fully-*banned* state on layer X. In contrast, regarding the destruction of layers, the navigation filter will remove layer X when both of the following conditions are satisfied: (1) the robot's heading should point to the target; in addition, (2) the space of directions associated with layer X should be less than half *banned*.

In order to clarify the previously stated layer-removal criterion, consider figure 3.8(b) at step 6, where layer 2 is created (hence, $X = 2$). From that moment on, only steps 10 and 14 comply with condition 1 since, in these steps, the precise angle that the robot is facing —as given by θ_{sol}— coincides with θ_{target}. Alternatively, relating to the second condition claimed, step 6, and steps 11 to 14, visibly fulfill it. In short, under $X = 2$, conditions 1 and 2 are jointly met at step 14 for the first time. Finally, it is important to remark that, after removing layer $X = 2$ at step 14 (consequently, now $X = X - 1 = 1$), the navigation filter continues following the contour of the obstacle on the basis of layer $X = 1$, as made clear by step 15.

As can be deduced from above, in the face of a multi-loop spiral-shaped obstacle, the navigation filter, by applying the multi-layer feature, will create as many layers as loops the spiral has. Each layer will be associated to a different loop; moreover, layer Y ($1 \leq Y \leq X$) will contain the obstacle information locally collected while traversing —in counterclockwise direction— the Y-th —from outer to inner— loop of the spiral-like obstacle. Taking advantage of all this information, the navigation filter will revert every loop traversal back —i.e. in clockwise direction—, with the aim of certainly avoiding the underlying *nesting* potential deadlock situation (in figure 3.8(b), this way of acting is revealed by steps 1 to 15, as well as by the subsequent straight-line segments connecting step 15 with the target point, which represent the rest of resultant robot's trajectory).

To conclude, let us explicitly put in words the trivial fact that the navigation filter knows that the robot is currently circumnavigating the boundary of an obstacle with *nesting* spiral-like shape when there exists more than one layer —or equivalently, when $X > 1$ $\Big($to this respect, notice additionally that any *nesting* potential deadlock situation becomes *simple* when, due to the removal of layers, X reaches its default value of 1, just as occurring at step 15 in figure 3.8(b)$\Big)$.

Summarizing, as a handy principle for decreasing / increasing the chances of cyclic behaviors / global convergence, the navigation filter, under the detection of either a *simple* or a *nesting* potential deadlock situation, does oblige the robot to keep performing the contour following process on the corresponding obstacle.

[5] As a special case, it is important to note that there is not early obstacle information at step 6. Moreover, in this step, layer 2 is exclusively feeded with the new data that 'overfull' layer 1

As a final point, algorithm 3.1 describes, step by step, how the navigation filter operates (observe that, with the main aim of making algorithm 3.1 reasonably clear, the pseudo-code associated with the handling —i.e. identification and avoidance— of *nesting*-type potential deadlock situations has been wholly omitted).

3.2 The Classical Potential Fields Method with no Local Minima

Broadly speaking, along this section, the *Potential Fields Method (PFM)* will be extended to properly avoid those trapping situations caused by the *local minima* problem. Specifically, such an improvement will be accomplished by merely incorporating the previously developed *navigation filter* into the typical control architecture of a *PFM*. As a final part of this section, the proposed *PFM*-based strategy will be widely tested in both simulated and real scenarios with the intention of clearly demonstrating the achievement of the local minima-free property. What is more, the aforesaid experimentation will also serve to show the appearance of two additional properties, not found in the original / classical *PFM*, which namely are: on the one hand, the capacity for successfully solving complex navigation tasks even when the detection of obstacles relies on low-cost sensors; and, on the other hand, the capability of producing, in general, shorter trajectories in comparison to other well-known techniques in the currently-concerned field of purely reactive navigation.

Next, before going into the description and the experimental evaluation of the proposal, two alternative formulations of the potential fields method will be deeply discussed. Concisely, the first formulation corresponds to the one of the classical approach put forward in [8] —and also briefly introduced in section 2.1—, while the second formulation is new and constitutes a further development of the previous one, in the sense of producing smoother and safer paths to the target —but without preventing the robot from being trapped into a local minimum, just like occurs with the classical formulation. Table 3.2 brings together the formal terms that will be shared by both formulations.

Finally, regarding the above, it is important to point out that one of these two formulations of a *PFM* —to be exact, the second one because of its distinctive advantage of generating better trajectories in terms of smoothness and obstacle clearance— will be later used in this section for building, in combination with the navigation filter, the intended *PFM*-based strategy with no local minima.

3.2.1 Going Deeply into the Classical Potential Fields Method

The classical potential fields approach computes the motion of the robot on the basis of two simple behaviors: *GoTo* and *AvoidObstacles*. More precisely, the former generates an attractive force in direction to the target, while the latter considers obstacles as repulsive surfaces. The robot follows the negative gradient of the resulting potential field towards its minimum, whose position is expected to coincide with the target point. In the following lines, this method is initially formalized for the case of one obstacle, being, afterwards, generalized.

3.2.1.1 Considering a Single-Obstacle Scenario

The artificial potential function applied to the robot has the form shown in equation 3.1, where $U_a(X)$ and $U_{r,o}(X)$ denote, respectively, the attractive / repulsive potential field induced by the target / obstacle in X. The attractive potential field is, on the other hand, defined according

Algorithm 3.1 Steps of the navigation filter at each control loop iteration

{For a greater clarity, the scope of the variables DeadlockSituation, NewObstacle, and CFD —the latter stands for Contour Following Direction— is considered to be *global*. Additionally, observe that both DeadlockSituation and NewObstacle denote a boolean-type variable that is globally initialized to false and true, respectively}

Get the up-to-date value of θ_{target} {Input}

{New state for the space of directions}
Acquire the local data provided by the onboard robot's sensors for obstacle detection

Merge the new obstacle information with that one previously collected

Compute the state —*banned* / *allowed*— of the angular regions based on the current robot's position

{Recognizing the beginning and the end of a *simple*-type potential deadlock situation}
if an almost fully *banned* space is detected **then**
  DeadlockSituation = true;
else if the current robot's heading / orientation matches θ_{target} **then**
  DeadlockSituation = false;
end if

if θ_{target} lies in an *allowed* region **and** DeadlockSituation is false **then**
  {The robot directly goes to the target point}
  $\theta_{sol} = \theta_{target}$;
  Remove, if any, all the obstacle information {This step resets the navigation filter}
  NewObstacle = true; {In subsequent iterations, a new obstacle may be found}
else {The robot follows the boundary of an obstacle}
  {Dealing with *simple*-type potential deadlock situations}
  if θ_{target} lies in an *allowed* region **then**
    {The robot should be forced to continue following the contour of the obstacle. To this end, the intended direction of motion of the robot, typically defined by θ_{target}, is now supposed to be given by θ_{banned}, which represents one of the directions in space whose state is *banned*}
    Look for a *banned* region in the space of directions
    Set the midpoint direction of the region resulting from the preceding search as θ_{banned}
  end if
  {Concerning the *Traversability* principle}
  if θ_{target} lies in an *allowed* region **then**
    Obtain for θ_{banned} the two alternative motion directions labeled as *left* and *right*
  else
    Obtain for θ_{target} the two alternative motion directions labeled as *left* and *right*
  end if
  {Concerning the *Tenacity* principle}
  if NewObstacle is true **then**
    Choose either *left* or *right* by applying a *fixed-beforehand* / *random* criterion
    Modify the variable θ_{sol} accordingly
    Keep the label of the direction which has been finally selected in CFD
    NewObstacle = false;
  else
    Choose either *left* or *right* in accordance with the content of CFD
    Modify the variable θ_{sol} accordingly
  end if
  {Removing progressively the information about obstacles which is no longer useful to navigate}
  Select the *banned* region next to θ_{sol} {Henceforth, such a region is referred to as r_{check}}
  if no obstacles are locally detected in the range of directions linked to r_{check} **then**
    {Notice that here, the term *locally* means the exclusive use of the robot's sensors}
    Change the state of r_{check} to *allowed*
    Forget those obstacles whose location led to the prohibition of r_{check}
    Set the midpoint direction of r_{check} as θ_{sol}
  end if
end if
return θ_{sol}; {Output}

Table 3.2: Terminology involved in the formalization of the potential fields method.

Term			Meaning
X	$=$	(x, y)	Current robot position
X_t	$=$	(x_t, y_t)	Target point
X_o	$=$	(x_o, y_o)	Location of a certain obstacle O
d_t	$=$	$\sqrt{(x - x_t)^2 + (y - y_t)^2}$	Distance to the target
d_o	$=$	$\sqrt{(x - x_o)^2 + (y - y_o)^2}$	Distance to the obstacle
$-\frac{\partial d_t}{\partial X}$	$=$	$-\left(\frac{(x-x_t)}{d_t}, \frac{(y-y_t)}{d_t}\right)$	Unit vector determining the direction of the attractive force
$\frac{\partial d_o}{\partial X}$	$=$	$\left(\frac{(x-x_o)}{d_o}, \frac{(y-y_o)}{d_o}\right)$	Unit vector determining the direction of the repulsive force
K_a, K_r			Gain factors for the attractive and repulsive forces

to equation 3.2. As can be observed, it is a positive quadratic function whose first derivative is continuous and its only minimum is located at $X = X_t$. The corresponding force expression derived from $U_a(X)$ is also given by equation 3.3. As for the repulsive potential field, it was selected so as to keep the resultant potential function $U(X)$ positive, continuous and derivable with a minimum at the target point. The details for $U_{r,o}(X)$ are found in equation 3.4. Notice that the influence of the obstacle is restricted to a neighborhood by means of the constant d_{max}. In this way, it is intended to prevent the alteration of the global minimum defined by the attractive potential function. Finally, equation 3.5 shows how the repulsive force is calculated.

$$U(X) = U_a(X) + U_{r,o}(X). \tag{3.1}$$

$$U_a(X) = \frac{1}{2}\, K_a\, d_t^2. \tag{3.2}$$

$$F_a(X) = -\vec{\nabla} U_a(X) = -K_a\, d_t\, \frac{\partial d_t}{\partial X}. \tag{3.3}$$

$$U_{r,o}(X) = \begin{cases} \text{if } d_o < d_{max} \\ \quad \frac{1}{2}K_r\left(\frac{1}{d_o} - \frac{1}{d_{max}}\right)^2, \\ \\ \text{otherwise} \\ \quad 0. \end{cases} \tag{3.4}$$

$$F_{r,o} = -\vec{\nabla}U_{r,o}(X) = \begin{cases} \text{if } d_o < d_{max} \\ \quad K_r\left(\frac{1}{d_o} - \frac{1}{d_{max}}\right)\frac{1}{d_o^2}\ \frac{\partial d_o}{\partial X}, \\ \\ \text{otherwise} \\ \quad 0. \end{cases} \tag{3.5}$$

3.2.1.2 The General Case

Once both $F_a(X)$ and $F_{r,o}(X)$ have been characterized, the computation of the latter will be generalized for a set of obstacles surrounding the robot. Assuming that the number of obstacles which have been detected by the robot's sensors is n, the resultant repulsive force $\Big(F_r(X)\Big)$ is simply determined by equation 3.6. Such a force is, afterwards, added to the attractive one in order to obtain the output of the method (see equation 3.7). Finally, this output should be translated into a control command through an appropriate algorithm. To this end, several control techniques are possible according to, mainly, the robot type, its kinematic constraints as well as the control choice.

$$F_r(X) = \sum_{i=1}^{n} F_{r,o_i}(X). \tag{3.6}$$

$$F(X) = F_a(X) + F_r(X). \tag{3.7}$$

3.2.2 A New PFM-type Formulation for Generating Smoother and Safer Trajectories

Our new formulation of the potential fields method computes the motion of the robot by means of a *GoTo* and an *AvoidObstacles* behavior just like the classical approach. However, there are important differences with the latter regarding the way how these behaviors are defined and their responses are coordinated in order to generate the control method's output. As a result of these changes, the trajectory of the robot is smoothed and the risk of collision with obstacles is almost negligible. Next, a description of both behaviors together with their corresponding coordination mechanism is given. Additionally, several results obtained by simulation showing the above-mentioned properties of the new formulation are finally presented.

3.2.2.1 The *GoTo* Behavior

By analyzing the role of the *GoTo* behavior in the context of the potential fields approach, one can easily guess that it is fundamentally responsible for the definition of an ideal path to the target point. This path is, afterwards, slightly modified by taking into account the

particular features of the environment —obstacles— where the robot is navigating. These changes, which are specifically carried out by the *AvoidObstacles* behavior, generally result in the loss of the path optimality from the point of view of its length, smoothness and danger[6]. In short, it seems clear that there is not a significant influence of the *GoTo* behavior on the causes of such lack of optimality in the trajectory followed by the robot. Consequently, new formulations for this behavior will not lead to relevant path improvements in the sense previously stated. Therefore, our effort should be focused on the rest of components of the control system.

Bearing in mind the previous discussion, an existing formulation suggested in [62] has been adopted for the *GoTo* behavior. This formulation possesses some differences with respect to the one originally put forward in the classical approach. Specifically, the attractive potential field which is defined produces, contrary to the latter, a force whose intensity is not proportional to the distance between the robot and the target point. The potential function, to be exact, has a quadratic behavior at the target neighborhood and an asymptotic one away from it. In this way, the robot is intended to behave in the same way with regard to a given obstacle configuration irrespective of its proximity to the target. Equations 3.8 and 3.9 show, respectively, the expressions for $U_a(X)$ and $F_a(X)$ where R is a positive constant value characterized by equation 3.10. Finally, in this last equation, F_{ref} and d_{ref} are two parameters which have to be set. To this end, notice that the former determines the norm of the force $F_a(X)$ precisely at distance d_{ref} ($d_t = d_{ref}$). The typical shapes of both $U_a(X)$ and $\parallel F_a(X) \parallel$ are depicted in figure 3.9.

$$U_a(X) = \sqrt{d_t^2 + R^2}. \tag{3.8}$$

$$F_a(X) = -\vec{\nabla} U_a(X) = -\frac{d_t}{\sqrt{d_t^2 + R^2}} \frac{\partial d_t}{\partial X}. \tag{3.9}$$

$$R = d_{ref} \sqrt{\frac{1}{F_{ref}^2} - 1}. \tag{3.10}$$

3.2.2.2 The *AvoidObstacles* Behavior

Most of the proposed repulsive potential fields appearing in the literature depend exclusively on the distance to the obstacles. In consequence, any obstacle has influence on the robot motion even if it is moving in a parallel direction, which, in general, leads to more irregular trajectories. A repulsive potential function that modifies the intensity of the force according to the relative angle (α) between the robot heading ($\vec{v}_\theta$) and the obstacle position is put forward next to counteract this effect $\Big($see figure 3.10(a)$\Big)$.

As can be observed in figure 3.10(b), two regions are defined around the robot, determining, each of them, a different way of computing the force intensity. They are called *influence* and *safety* areas. Regarding the former, it is intended to limit the influence of the obstacles on the robot motion to those inside the region. Its shape, which is elliptical, pays special

[6]Notice that the closer to the obstacles, the riskier is the trajectory

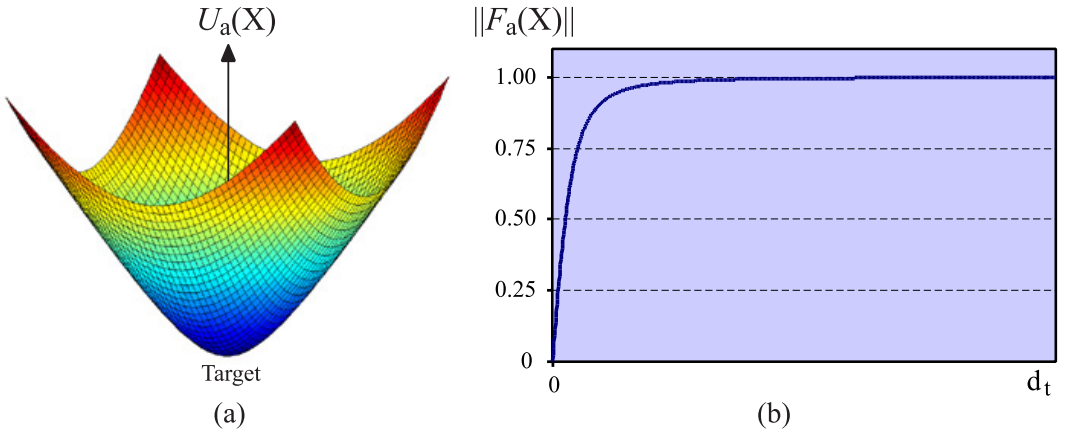


Figure 3.9: The *GoTo* behavior: (a) attractive potential field; (b) norm of the derived force.

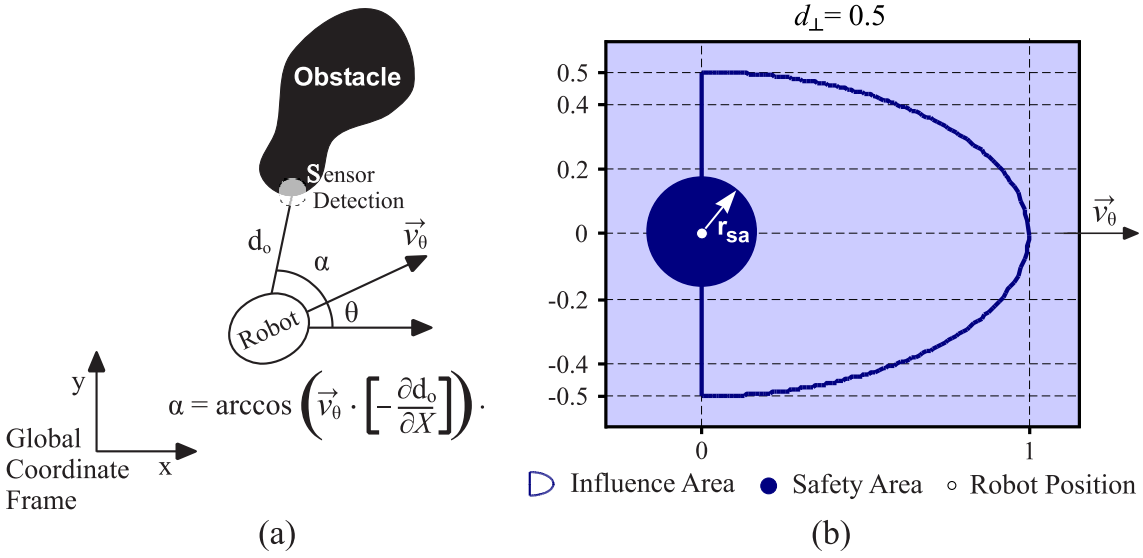


Figure 3.10: The α-dependent repulsive potential field: (a) computation of the angle α; (b) normalized *influence* and *safety* areas.

attention to obstacles located in front of the robot due to the greater difficulty to avoid them. Equation 3.11 expresses the distance from the robot position to the boundary of the region as a function of α, representing $d_\perp$ the normalized length of the semiminor axis of the ellipse $\left(d_\perp = d_{max}(\frac{\pi}{2})/d_{sr},\ 0 \leq d_\perp \leq 1\right)$ and d_{sr} the maximum range of the on-board sensors used for obstacle detection. Finally, notice that the condition which has been added to the equation prevents the robot from considering obstacles behind itself.

$$d_{max}(\alpha) = \begin{cases} \text{if } 0 \leq \alpha \leq \frac{\pi}{2} \\ \qquad \left(\sqrt{\left(1 - \frac{2\alpha}{\pi}\right)(1 - d_\perp^2) + d_\perp^2}\right) d_{sr}, \\ \\ \text{otherwise} \\ \qquad 0. \end{cases} \tag{3.11}$$

The force intensity in the *influence area* is initially evaluated on the basis of the distance d_o between the robot and the obstacle. Afterwards, the resultant value is weighted according to a cosine function. As a result, the robot experiences the repulsive force at its full magnitude when it frontally approaches the obstacle ($\alpha = 0$). On the contrary, as the robot turns towards a direction alongside the obstacle's boundary the force is weakened, achieving its minimum intensity with $\alpha = \frac{\pi}{2}$.

As for the *safety area*, it is characterized by a circular shape of radius r_{sa} and is intended to counteract an immediate risk of collision. Under these circumstances, a quick response is required irrespective of the angle α. Hence, the force is computed following the classical method where only d_o is taken into account.

Equation 3.12, in line with the preceding descriptions, provides the force expression $F_r^\alpha(X)$ for an obstacle o, where $w_\perp$ ($0 \leq w_\perp \leq 1$) represents the weakening factor of a force for $\alpha = \frac{\pi}{2}$ and the term F_{min} ($0 \leq F_{min} \leq 1$) denotes the minimum force intensity before the α-weighting, if applied, when the obstacle is detected in either the *safety* or the *influence* area.

$$F_{r,o}^\alpha(X) = \begin{cases} \text{if } d_o \leq r_{sa} \\ \qquad \left(1 - (1 - F_{min})\frac{d_o}{d_{max}(0)}\right) \frac{\partial d_o}{\partial X}, \\ \\ \text{else if } d_o \leq d_{max}(\alpha) \\ \qquad \left(1 - (1 - F_{min})\frac{d_o}{d_{max}(0)}\right) \cos\left(\frac{2\alpha}{\pi} \arccos(w_\perp)\right) \frac{\partial d_o}{\partial X}, \\ \\ \text{otherwise} \\ \qquad 0. \end{cases} \tag{3.12}$$

To conclude, it is important to note that a force $F_{r,o}^\alpha(X)$ has to be computed for each of the obstacles locally surrounding the robot. To this end, our *AvoidObstacles* behavior internally keeps a local occupancy grid C, whose location is such that the vehicle is always at its center. As the robot moves around, range readings are taken and projected into the grid obtaining, for each cell $C[i, j]$, its corresponding probability of occupancy. According to this information, the presence of obstacles is probabilistically determined in a very easy way. More exactly, a repulsive force is generated for those cells of the grid whose occupancy probability is higher than a user-definable threshold. The resultant set of forces is, later, sent to a coordination mechanism which will be explained next.

3.2.2.3 Coordinating their Responses

In the generic context of reactive control systems, coordination mechanisms are typically classified into two groups: competitive and cooperative. In the former, the control of the robot is exclusively given to one behavior until the next execution cycle. On the contrary, the latter combines recommendations from multiple behaviors to form a single control action which represents their consensus. Both strategies offer some advantages and disadvantages. On the one hand, competitive methods provide good robustness[7] but non-optimal paths from the point of view of their smoothness and length. Cooperative methods, on the other hand, present opposite features. In short, it seems obvious that a hybrid methodology that is able to make the most of both coordination strategies is desirable. With this purpose, the general idea of a new hybrid coordinator will be described in the following, being, afterwards, formalized for a specific case.

The intensity of a force represents, to a certain extent, the urgency of the generating behavior for taking the control of the robot at a precise moment. The hybrid coordination mechanism that is proposed exploits this fact to dynamically assign a priority to each of the behaviors' responses —remember that the *AvoidObstacles* behavior may generate multiple responses / repulsive forces. Two different levels of priority generically called *low* and *high* are defined. On the other hand, an attractive / repulsive force is considered to be of *low / high* priority when its intensity, whose value is bounded to the real interval [0,1] (see equations 3.9 and 3.12 again), is below / above a user-definable threshold. It is important to highlight that this threshold can be differently set for each behavior. Once the forces have been properly classified, the coordinator acts competitively between priority levels and cooperatively inside them in order to obtain the final control system response $F^{\alpha}(X)$.

Equation 3.13 formalizes the previous idea with the aim of favoring the robot's safety. Notice that the α-dependent repulsive forces are now denoted as $F^{\alpha}_{r,c[i,j]}(X)$ in accordance with the occupancy grid used by the *AvoidObstacles* behavior. Moreover, in the expression, the function $O(i,j)$ returns 1 when the occupancy probability of the cell $C[i,j]$ is high enough so as to assume the presence of an obstacle in that position —0 otherwise. On the other hand, $d(i,j)$ is equivalent to d_o. As can be observed, the control of the robot is exclusively given to the *AvoidObstacles* behavior in situations where at least one obstacle is detected inside the *safety* area. In the rest of cases, the weighted addition of the attractive and repulsive forces is actually performed.

As a final point, let us remark that equation 3.13 inherently supposes that the priority thresholds associated with the *GoTo* and the *AvoidObstacles* behaviors have a value of 1 and $\left(1-(1-F_{min})\frac{r_{sa}}{d_{max}(0)}\right)$, respectively.

$$F^{\alpha}(X) = \begin{cases} \text{if } \exists i,j \;\mid\; d(i,j) \leq r_{sa} \text{ and } O(i,j)=1 \\ \qquad \displaystyle\sum_{i,j \;\mid\; d(i,j)\leq r_{sa}} \left(F^{\alpha}_{r,c[i,j]}(X)\; O(i,j)\right), \\ \text{otherwise} \\ \qquad K_a F_a(X) + K_r \displaystyle\sum_{i,j} \left(F^{\alpha}_{r,c[i,j]}(X)\; O(i,j)\right). \end{cases} \tag{3.13}$$

[7] Owing to the fact that the active behavior with the highest priority, either static or dynamic, will always take control of the robot in competitive coordination mechanisms, a more focused response can be given to critical situations such as the detection of a very close obstacle

3.2.2.4 Identification of the Main Properties of the Proposal

The formulation which has been previously proposed for the potential fields method provides the control strategy with two new properties: (1) the smoothness of the robot's path is improved, and (2) the risk of collision with obstacles is also reduced. These properties have been clearly identified by comparing the results of our proposal with the ones obtained by the classical approach for which two different versions were contemplated in order to achieve a larger completeness. More precisely, the first version faithfully corresponds with the description given in section 3.2.1 where a cooperative coordination mechanism was employed to combine the responses of both the *GoTo* and *AvoidObstacles* behaviors (see equation 3.7). Distinctively, in the second version, the coordination of these two behaviors is competitive. To this respect, remember that this competition requires the assignment of a static priority for each behavior (as explained in section 1.3, this requirement arises because a competitive coordination technique acts by selecting as output the response of the active behavior with the highest priority). In the particular case considered here, the competitive version of the classical approach prioritizes the *AvoidObstacles* behavior over the *GoTo* behavior with the aim of increasing safety.

Under this comparative framework, two experiments were carried out on $NEMO_{CAT}$ [63] by using the simulated robot GARBI (refer to appendices A and B for details about, respectively, the underwater vehicle GARBI, and the simulator $NEMO_{CAT}$ which has been actually developed during this dissertation). Figure 3.11(a) shows the results for the first experiment where the robot had to overcome some obstacles progressively narrowing the free space available for navigating towards the target. As can be plainly observed, with our formulation, the path of the robot turned out to be much smoother. On the other hand, paying attention on safety concerns, the second experiment was intended to reveal the reluctance of the vehicle to collide when it was trapped into a box-shaped canyon and the target point was located on the other side of the obstacle walls $\Big($look at figure 3.11(b) for a precise understanding of the environment set up$\Big)$. With this purpose in mind, several simulations were conducted, each of them being characterized by a different setting of the gain linked to the *GoTo* behavior. To be more exact, such a gain —K_a— was ranged from 0.05 to 2.00 in 0.05 steps, resulting thus in forty simulations (in contrast, notice that the gain of the *AvoidObstacles* behavior —K_r— was always fixed to 1.00). The results coming from all these simulations are jointly plotted in figure 3.11(c). Regarding this figure, it is important to highlight that no results are presented for the competitive version of the classical approach. This fact is due to the particular context of the experiment which exploits the concept of gain. As it is well known, a *PFM* that adopts a competitive coordination mechanism gets rid of this concept because of computing its output by choosing the response of a single behavior, and not by merging the whole set of behavioral responses. In summary, the lack of the gain concept does not permit including the competitive version of the classical approach in the comparative study of figure 3.11(c). Despite this, there is no doubt that the robot would never collide when navigating competitively inside the box-shaped canyon. To conclude, by deeply examining the results of figure 3.11(c), it seems evident that our proposal guarantees the robot's safety against collisions.

3.2.3 The Navigation Filter as a part of the Potential Fields Method

Figure 3.12 illustrates the integration of the navigation filter into the new formulation of the *PFM* previously presented. As can be observed in such a scheme, the navigation filter

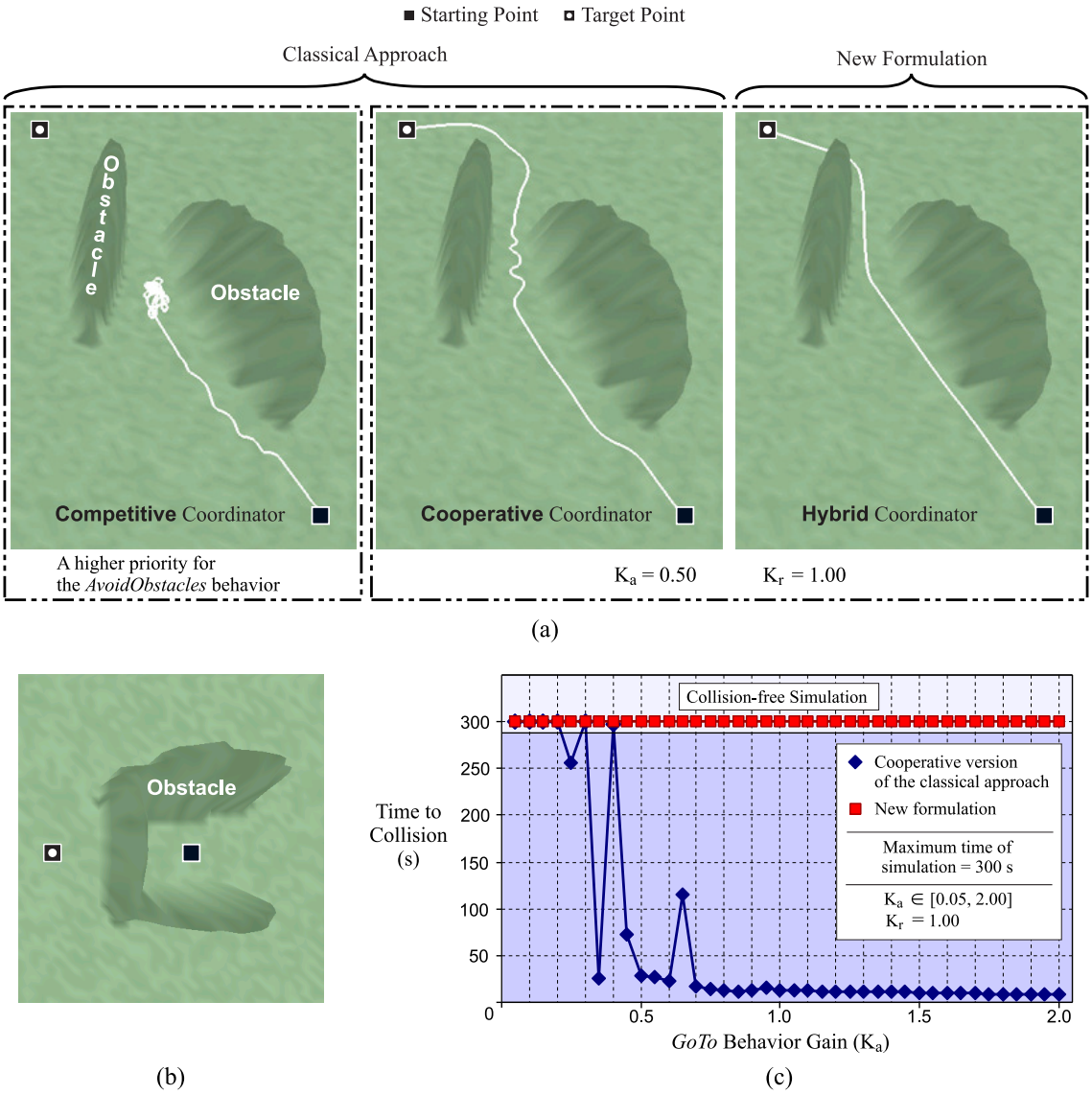


Figure 3.11: Comparison between the classical and the suggested formulation of a *PFM* from two points of view: (a) smoothness of the robot's trajectory; (c) robot's safety (besides, (b) illustrates the scenario where the safety-assessment experiment was performed). In (a), relating to both the cooperative and hybrid coordinators, observe that the contribution to the output motion vector of the attractive and repulsive forces was specifically determined by the gain factors $K_a = 0.50$ and $K_r = 1.00$, respectively (in that regard, recall that the hybrid coordinador takes into account these gains only when behaving cooperatively, i.e. when not finding obstacles inside the so-called *safety* area).

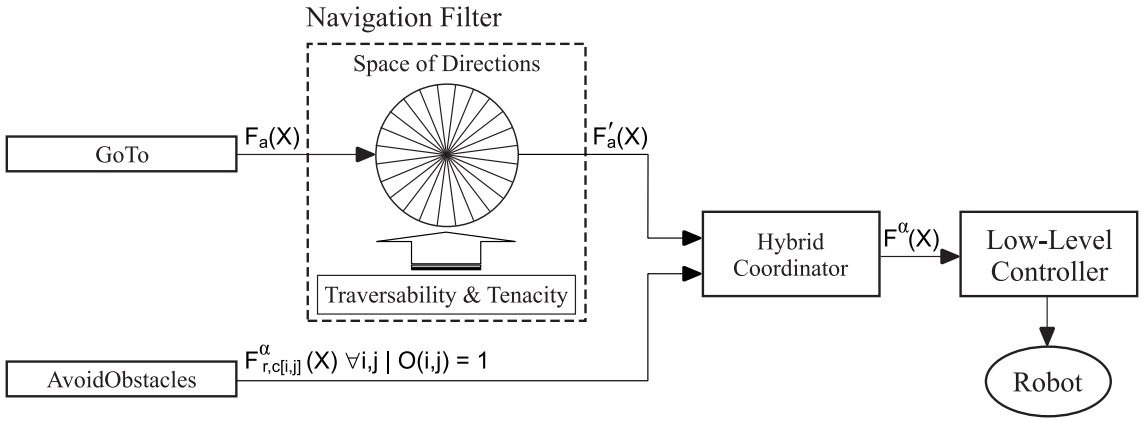


Figure 3.12: Using the navigation filter for building a local minima-free *PFM*.

is used in a way to change the attractive-type force that the hybrid coordinator receives as input; specifically, the original input $F_a(X)$ —as given by equation 3.9— is now replaced by $F'_a(X)$. Comparing $F_a(X)$ and $F'_a(X)$, it is important to note that these force vectors have the same magnitude $\Big(\text{therefore, } |F_a(X)| = |F'_a(X)|\Big)$, but a generally different direction. In that last respect, the direction of $F_a(X)$ is θ_{target}, while θ_{sol} corresponds to the pointing direction of $F'_a(X)$. Being brief, θ_{sol} —which is calculated on the basis of algorithm 3.1— drives the robot towards the target location, just like θ_{target}, but with the additional advantage of avoiding any local minimum in the potential field.

3.2.4 Experimental Evaluation of the Absence of Influence of Local Minima

The purpose of this section is two-fold: on the one hand, to experimentally evaluate the absence of the local minima problem when navigating according to the new *PFM*-based approach put forward in figure 3.12; and, on the other hand, to provide a comparison of such a novel approach with other purely reactive strategies in terms of both mission completeness and path length performance.

Next, first of all, the proposed strategy is demonstrated to achieve successful navigation in several real scenarios containing obstacles typically causing local minima. Moreover, in order to increase difficulty, these experiments are deliberately carried out by using a robot equipped with sensors which offer noisy perceptions of the environment as well as just a partial coverage of the local robot's surroundings. After that, and closing this section, the comparative study pointed out above is performed under simulation.

Before continuing with the intended experimentation, some discussion is needed to clarify a functional aspect of our proposal that has been left partially open-ended so far. To this respect, in section 3.1.4, three alternative criteria were suggested to choose the direction —either left or right— to follow the contour of the detected obstacles. By analyzing both the advantages and disadvantages of these criteria, one of them —named *minimum turn*— was definitely discarded, while the other two —named *fixed beforehand* and *random*— were regarded, to some extent, as complementary since they do give different trade-offs between mission completeness and path length performance. Broadly speaking, the *random* criterion, as opposed to the *fixed-beforehand* criterion, augments the chances of ending up reaching the target, as well as of producing longer trajectories. At this point, it is important to remark that the real and simulated experiments to be presented involve a set of scenarios where

convergence could be indistinctively accomplished by any of the two latter-referenced criteria. In view of that, it seems clear that the decision of adopting either the *fixed-beforehand* criterion or the *random* criterion should exclusively depend on performance issues. In short, and contrary to what one might expect (essentially, because of selecting the choice that favors the generation of longer / worse trajectories), the criterion finally adopted has been *random*. The main reason for such a decision is to place the whole experimentation in a worst-case context with the aim of helping us to draw more relevant conclusions from the results obtained. As a final comment, notice that our proposal will be referred to as *Random* T^2 hereafter.

3.2.4.1 Tests with a Real Robot

The strategy *Random* T^2 was tested in two scenarios with a local minimum by using the real robot SoccerBot (refer to appendix A for a complete description about this robot). Only three infrared —IR— sensors distributed to the left, right, and at the front of SoccerBot were employed to measure the distances to the obstacles. Figures 3.13 and 3.14 depict the resultant robot's trajectory for each of the missions considered. As can be observed, the first experiment consisted of a typical U-shaped obstacle, while the second experiment was a slight alteration of the former where one of its ends was extended to form what has been called a *simple* potential deadlock situation (see section 3.1.4 for an explanation of the different types of potential deadlock situations —merely, *simple* and *nesting*). In both experiments, *Random* T^2 properly guided the robot SoccerBot to the target point, making thus clear the ability of this strategy to escape from local minima, even when navigating with low-cost sensors such as IRs.

3.2.4.2 An Extensive Comparative Study by Simulation

In this section, a study on the path length performance of the strategy *Random* T^2 is presented and deeply discussed. Moreover, as a key part of this study, *Random* T^2 is compared against five other algorithms from the related literature. The comparison is carried out in seven troublesome scenarios, which are simulated by using the software *MissionLab*.

Integration into *MissionLab*

MissionLab [64, 65, 66] is an open-source C++ suite of software tools based on the AuRA architecture [67]. From the standpoint of functionality, MissionLab allows the user / operator to define and execute missions using simulated or real robots, as is illustrated in figure 3.15. MissionLab has been developed and freely distributed[8] by the Mobile Robot Laboratory led by professor Ronald Arkin at Georgia Institute of Technology (Atlanta). Nowadays, this laboratory constitutes an outstanding and active research center in the field of reactive robotics. Many different reactive —and purely reactive— navigation algorithms have been proposed by members of the Arkin's lab during the last one-and-a-half decade. What is more, most of them have been incorporated into MissionLab by their own authors.

Getting to the point, MissionLab has been chosen here as the testbed on which to conduct the intended comparative study, with the clear aim of taking advantage of the wide set of state-of-the-art algorithms that such a software does include. To make this choice possible, nevertheless, the strategy *Random* T^2 had to be implemented on MissionLab. All details about this implementation can be found in [68].

[8]The latest version of MissionLab can be downloaded from http://www.cc.gatech.edu/aimosaic/robot-lab /research/MissionLab/

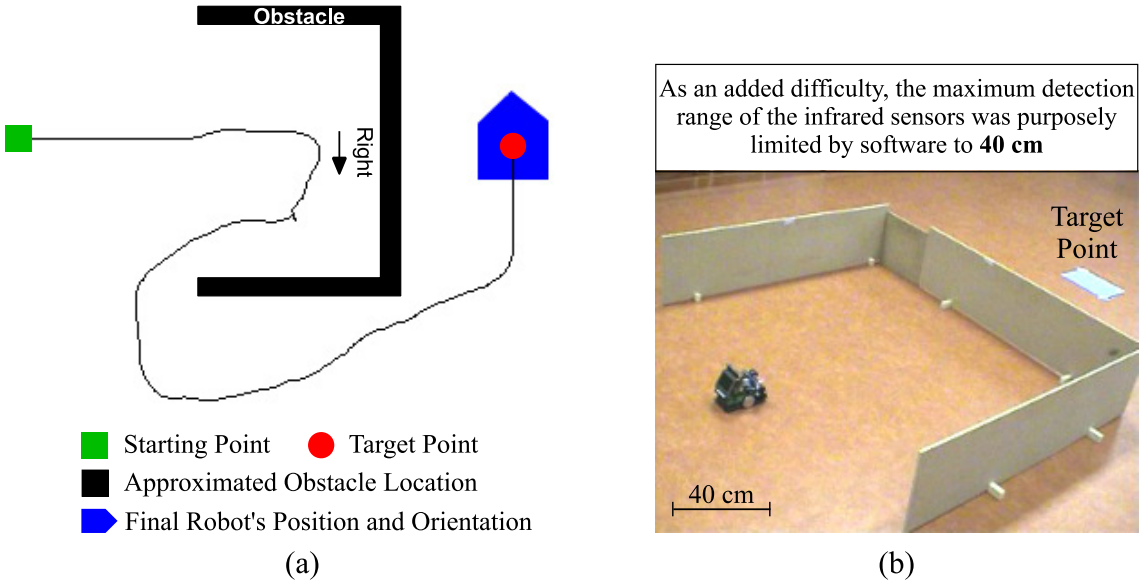


Figure 3.13: The robot SoccerBot escaping from a U-shaped obstacle: (a) trajectory generated according to the strategy *Random* T^2 (here, the following observations apply: on the one hand, the aforesaid strategy randomly decided to follow the boundary of the obstacle to the right; on the other hand, observe that there is an evident orientation error in the dead-reckoning estimation of the robot's path due to the Soccerbot's tendency to wheel slippage); (b) the environment set up built by means of several wood boards.

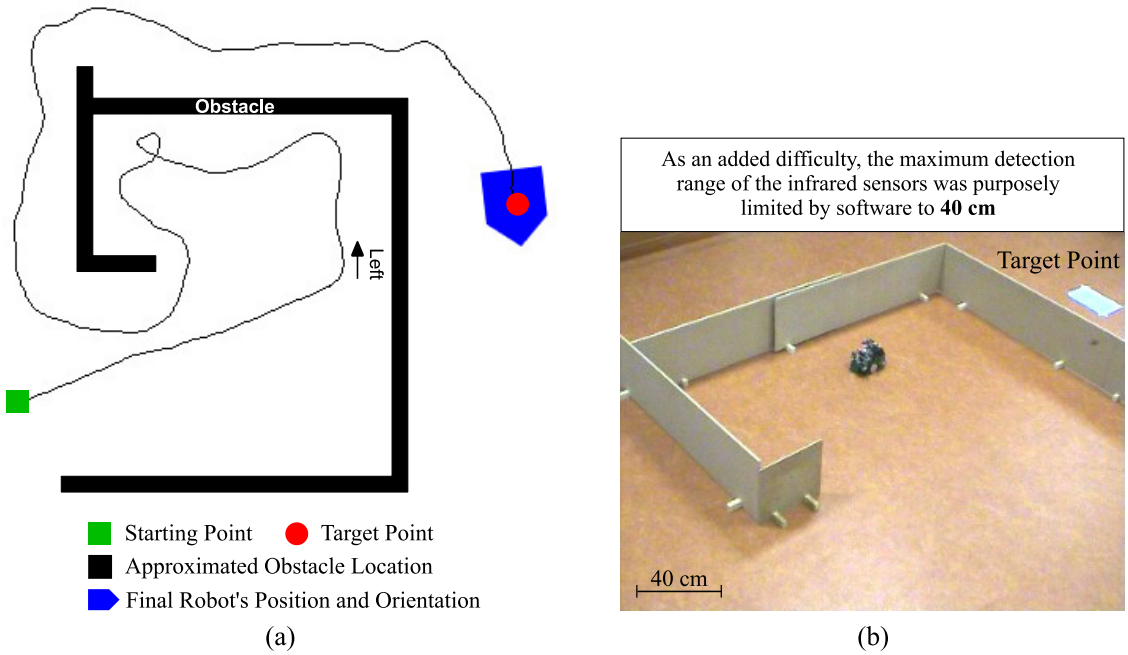


Figure 3.14: The robot SoccerBot overcoming a *simple*-type potential deadlock situation: (a) trajectory generated according to the strategy *Random* T^2 (observe that, on this occasion, the boundary of the obstacle is followed to the left); (b) a snapshot during the mission. The reader can find a ≪video≫ of this experiment in the electronic version of the document.

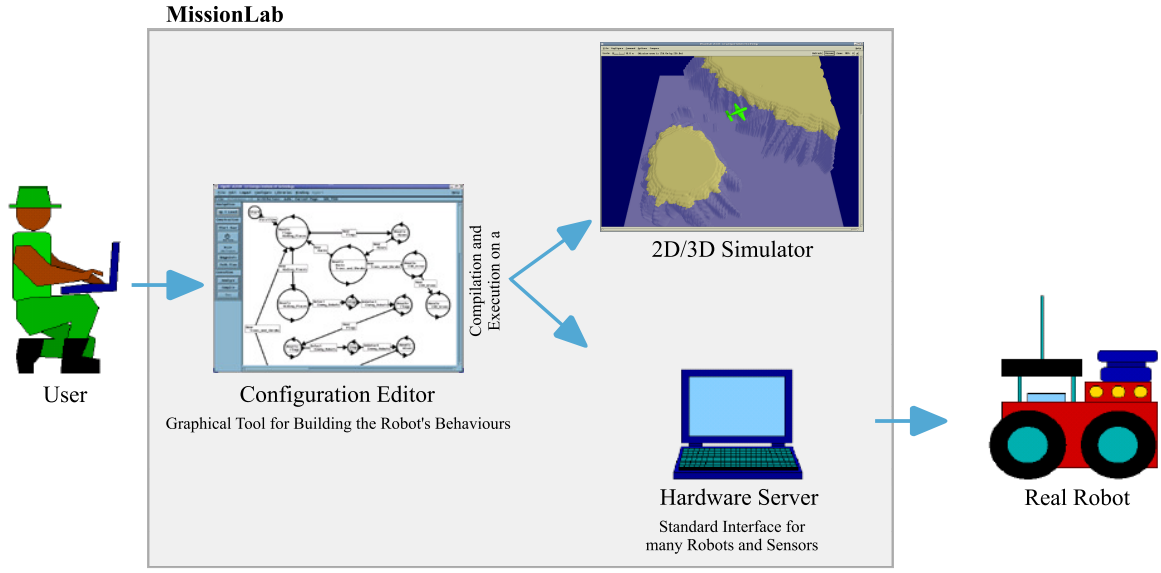


Figure 3.15: The interaction of a user with *MissionLab*.

Strategies under Consideration

Leaving aside the strategy *Random* T^2, several algorithms have been considered to take part in our comparative study. All of them were lengthily described in chapter 2. Despite this, their main features are pointed out next:

- *Avoiding the Past* [20]. The robot moves to the user-specified target point while being repelled from locations which were already visited. With this purpose, a local map of the environment implemented as a two-dimensional grid is stored in memory, where a different value is assigned to visited and non-visited locations. As the robot visits an area more times, the values of the corresponding cells in the grid increase and, consequently, the resultant repulsive force exerted by such cells increases as well. In this way, it is intended to favor the continuous exploration of new regions of the navigation environment avoiding thus, at least apparently, the robot gets stuck into a local minimum.

- *Learning Momentum* (*LM*) [29]. This strategy adjusts the behavioral parameters of a particular purely reactive control system at runtime instead of using static values. A module called *Adjuster* is precisely responsible for this task. The operation of this module is based on recent experience and a set of heuristic rules that identifies when good progress to the target is being made. According to this, the gains as well as other parameters of the three behaviors making up the control system —GoTo, AvoidObstacles, and Noise— are properly altered.

- *Micronavigation* (μNAV) [21]. This approach tries to solve the problem of autonomous robot navigation from a minimalist point of view by only using, as its authors say, a handful of bytes. Specifically, the robot is provided with a hierarchy of simple behaviors designed for smooth obstacle avoidance through the *equipotential line* concept and for escaping from concavities.

- A case-based reasoning (*CBR*) technique for the automatic selection and learning of behavioral parameters has also been taken into consideration [33]. At the start, an empty library of cases is available, where each case defines a complete set of parameters for the particular behavior-based control system used by the robot. New cases are created and optimized as the result of an automatic experimental procedure. On the other hand, their selection is based on the current environmental features. Notice that in order to obtain a good performance, a training stage is recommended to be carried out before trying to solve the desired navigation tasks. Once the training is over, the case library which has been learned can be employed with better chance of success.

- *Bug2* [53]. This is a well-known member of the family *Bug*, which is one of the most popular families of algorithms for path planning with incomplete information. The target achievement guarantee whenever possible is a key common characteristic of the Bug-like algorithms (refer to section 2.6 for further information).

  As for the specific strategy *Bug2*, two basic behaviors, GoTo and ContourFollowing, alternate the control of the robot. Initially, the GoTo behavior is active. Moreover, this behavior keeps active until the detection of an obstacle. At that moment, the ContourFollowing behavior starts a contour following process on the just detected obstacle. Such a process is left by the robot when it cuts the virtual line connecting the starting and the target points, also called *Main Line*.

Concerning the algorithms previously summarized, it is worth to explicitly mention that the first four of them do make the robot navigate towards the given target in a purely reactive manner, just like the strategy *Random* T^2. On the other hand, as compared to this first set of algorithms, the fifth —and last— method named *Bug2* involves a conceptually different way of doing navigation, which is also referred to as reactive but without the qualifying adverb 'purely'. Going further into this point, Bug2 is considered to be a reactive navigation scheme because of computing the resultant robot's path on the basis of exclusively local plans. In addition, within such a reactive context, Bug2 is not regarded as of pure type due to the fact that it keeps and uses certain global information —to be exact, the so-called *hit* points— in order to guarantee convergence to the target position.

Generally speaking, common sense suggests that the comparison of different types of techniques is inherently unfair. This situation, nevertheless, is going to be actually found in the present study when comparing the algorithm Bug2 against the strategy *Random* T^2 (in such a comparison, Bug2 will be clearly favored since, as described above, it employs richer —global— information about the navigation environment, which means more a priori chances to obtain better results in terms of path length performance). To this respect, notice that there is a special reason for the 'unfair' inclusion of the algorithm Bug2 in this comparative study. More exactly, such a reason obeys to the need imposed by a forthcoming chapter —specifically, by chapter 5— to identify the real, and not just the theoretically expected, relative advantages of Bug2 and *Random* T^2, with the ultimate intention of mixing both algorithms in a way that making the most of each of them. In closing, the comparison between the two aforesaid algorithms that our study is next going to provide will be exploited at a later stage.

Results for a Representative Set of Missions

Different tests of increasing complexity have been performed in *MissionLab*, simulating a holonomic robot equipped with several range finders and wheel encoders to calculate its position by means of dead-reckoning.

On the one hand, the first three missions were devised to show the main weaknesses of the *Avoiding the Past*, *LM*, and *CBR* algorithms versus *Random* T^2. As can be observed in figure 3.16, in the first mission, walls of different length impede the progress of the robot towards its target. The second environment, on the other hand, corresponds to a very deep box-shaped canyon. Finally, the third one represents, in a simplified way, an office where several rooms and corridors can be easily distinguished. It is important to highlight that this last mission was put forward in [69] to evaluate the behavior of a particular hybrid control system with deliberative capabilities. This fact allows getting a general idea of its difficulty. As for the results, none of the aforementioned algorithms with the exception of *Random* T^2 was able to successfully carry out the whole set of missions considered, which shows their poor effectiveness to escape from large trapping areas. These results were obtained by fixing a maximum time to complete a mission equal to twice the one of *Random* T^2. Once this time was over, the simulation was stopped. In the following, some specific comments are given for each algorithm:

- *Avoiding the Past*. Important difficulties have been encountered so as to configure the numerous parameters of this algorithm, since small modifications of their values have generally resulted in big changes of the robot's behavior. Figure 3.16(a) depicts how the robot solved two of the three missions proposed by applying the best set of parameters which was found. Specifically, the typical U-shaped obstacle was the only environment not overcome, which makes evident the inability of the approach to move the robot in a direction opposite to the target.

- *LM*. As was already pointed out in section 2.2.2, this strategy can be implemented in two different ways called *ballooning* and *squeezing*. More exactly, the former works better when facing obstacles such as box-shaped canyons, while the latter allows the robot to suitably navigate through environments built with small and closely spaced obstacles. Our choice based on the mission features was finally *ballooning* favoring thus the best *LM*'s performance. Despite this effort for an appropriate tuning, none of the missions was accomplished within the available time $\Big($look at figure 3.16(b)$\Big)$. In this regard, notice, nevertheless, that successful results could have been obtained for this algorithm by prolonging the corresponding simulations due to the existence of a wander / random behavior into its control architecture. The same reasons, on the other hand, do not permit establishing an upper bound to such extension, limiting thus the practical usefulness of the approach in complex scenarios. Lastly, as for the robot's trajectory, it has shown to be quite irregular in all the experiments as a result of the abrupt changes in the robot's heading caused by the above-mentioned inherent wandering process.

- *CBR*. MissionLab brings with it an empty library of cases for this learning unit. Under these circumstances, however, the performance of the whole control system is expected to be very low because of, precisely, such lack of prior knowledge. To avoid this problem, a preliminary training stage was carried out before obtaining the final results. This stage was intended to learn optimal behavioral parameterizations —that is to say, optimal cases— to overcome the most usual obstacle configurations that the robot might find. The same three missions previously explained were used in this training process, defining thus the *CBR*'s case library according to the specific navigation tasks to be solved. Figure 3.16(c) shows the path produced by the robot in each of the missions after a training of several hours. As can be observed, the target position was not reached on

Table 3.3: Comparing the path lengths of *Random* T^2 and μNAV. In the adopted notation, the term L_{C_i} represents the length of case i, that is, the length of one of the possible paths that may be generated by the strategy *Random* T^2 (the long-dash symbol is used in those missions where the total number of cases is less than four).

Mission	Algorithm Type					
	Random T^2					μNAV
	L_{C_A}	L_{C_B}	L_{C_C}	L_{C_D}	APL	
4	143.24	63.67	—	—	103.45	120.96
5	297.78	—	—	—	297.78	1364.13
6	522.93	558.66	548.53	582.93	553.26	1020.54
7	101.00	312.41	458.70	—	296.13	156.02
Total (m)					1250.62	2661.65

any occasion. Similar conclusions to those of *LM* can be drawn for *CBR* relating to both the possible late target achievement and the generation of erratic trajectories.

- *Random* T^2. As its name suggests, this is a non-deterministic strategy, which means that, at exactly the same scenario, different paths may be obtained in different runs of the MissionLab's simulator. In order to manage this outcome variability, a stochastic analysis on the average length of all possible paths that *Random* T^2 could generate in each of the seven missions involved in this comparative study was performed $\Big($refer to appendix C for details about the calculation of such an average path length $(APL)\Big)$. The results of this analysis for missions 1 to 3 can be found in figure 3.16(d).

Continuing the study, four extra missions were considered so as to evaluate and compare the performance of the μNAV algorithm against *Random* T^2 (notice that these missions are a representative subset of those appearing in [21], which were specifically designed for showing the navigation skills of the former / competing method). The results that were obtained through the simulations are graphically illustrated in figure 3.17, being also presented in tables 3.3 and 3.4 from the quantitative viewpoint of the length of the paths. As can be observed in table 3.4, the strategy *Random* T^2 produced, on average, trajectories between the starting and the target points 2.03 times shorter than μNAV. Broadly speaking, this difference in performance comes from the fact that μNAV allows the robot to head for the target as soon as it is faced without any immediate obstacle on its way, while *Random* T^2 limits the applicability of such a rule to situations where neither *simple* nor *nesting*-type concavities are detected.

To finish, the theoretical / ideal path of the algorithm *Bug2* for the whole set of missions earlier defined was drawn by hand in accordance with the steps described in procedure 4.1[9]. Figure 3.18 depicts such handy drawing paths, while tables 3.5 and 3.6 provide the comparative data with respect to our proposal. As indicated in table 3.6, *Bug2* generated, on average, trajectories 1.15 times longer than *Random* T^2. On this occasion, the strict condition linked to the end of the contour following process is the leading cause of the lower performance of *Bug2*. In this regard, remember that the robot should wait for the crossing of the so-called *m-line*

[9] Unfortunately, *Bug2* is not implemented on MissionLab so that no simulations could be carried out with it

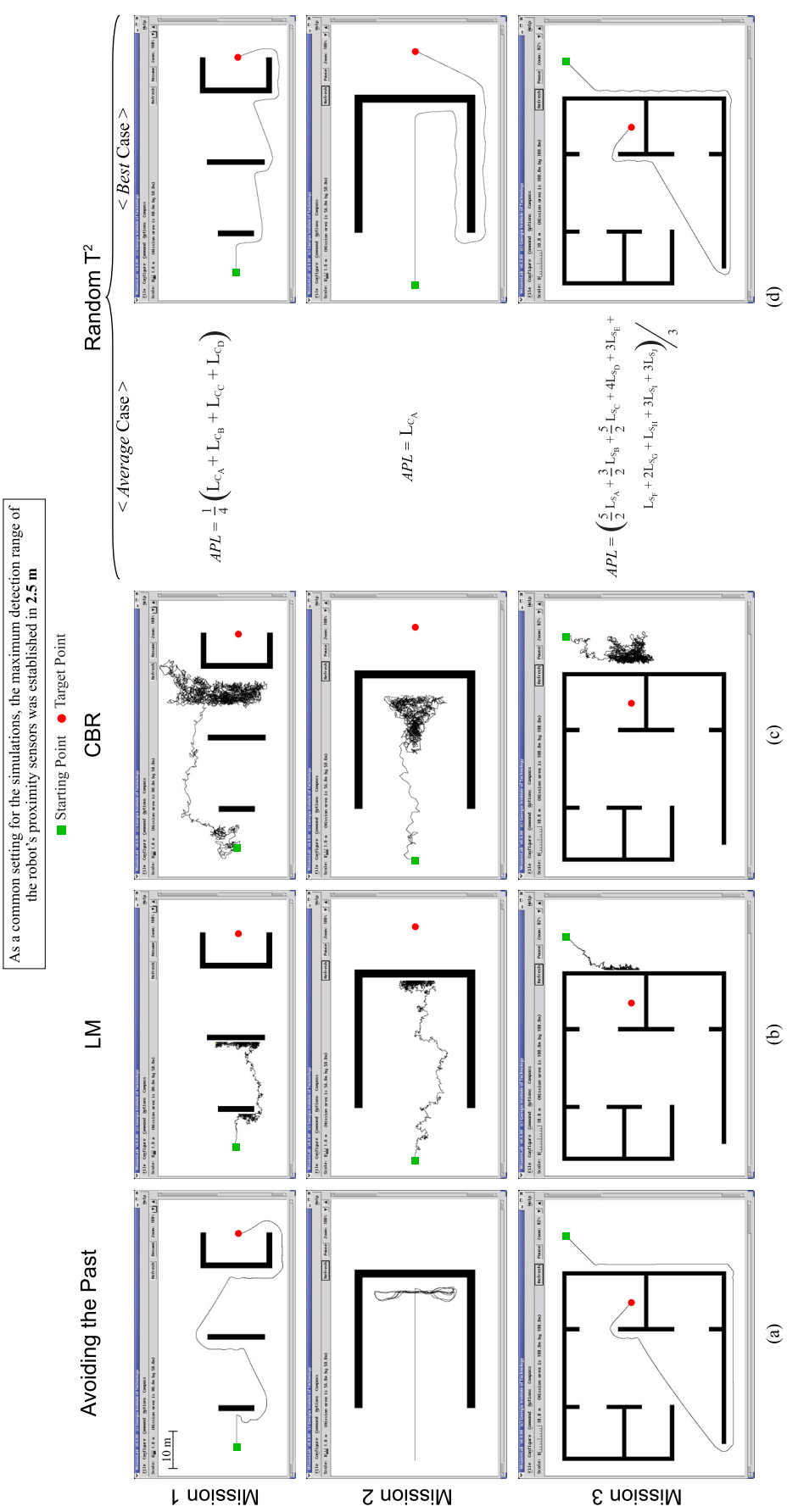


Figure 3.16: From left to right, trajectories generated by *Avoiding the Past*, *LM*, *CBR*, and *Random* T^2 in three troublesome scenarios (regarding the last-mentioned strategy, observe that, in addition to the expression of the average path length, the best *Random* T^2-based path is also displayed).

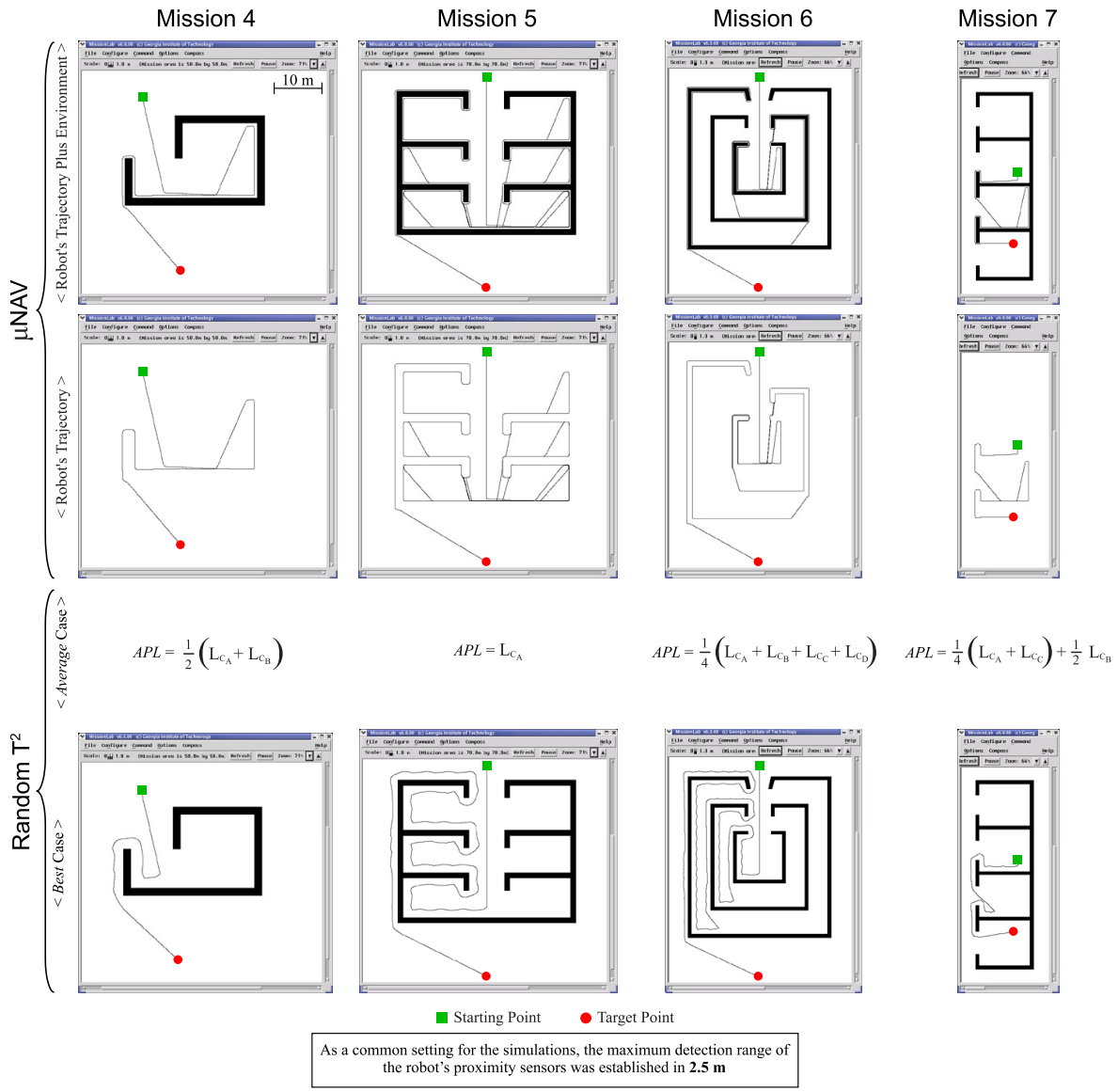


Figure 3.17: Confrontation of *μNAV* and *Random* T^2 in four scenarios where concavities appear forcing the robot to face the target point.

Table 3.4: Relative performance of *Random* T^2 and μNAV.

Mission	4	5	6	7	Average
$\frac{\mu NAV}{\text{Random } T^2}$	1.17	4.58	1.84	0.53[a]	**2.03**

[a]This result simply confirms a common-sense remark made by Vladimir J. Lumelsky in [54]. He essentially stated that, when comparing algorithms based on different principles, a scenario can be always constructed to exploit the strengths of just one of the algorithms under consideration. At this point, the reader should recall that missions 4 to 7 had been expressly designed for testing the μNAV strategy. Despite this, it is important to realize that, in mission 7 —which is the only unfavorable mission—, the best path found by *Random* T^2 was 1.54 times shorter than μNAV (see table 3.3 again).

Table 3.5: Comparing the path lengths of *Random* T^2 and *Bug2*.

Mission	Algorithm Type					
	Random T^2					*Bug2*
	L_{C_A}	L_{C_B}	L_{C_C}	L_{C_D}	Average	
1	88.56	98.26	109.52	99.48	98.96	123.00
2	113.78	—	—	—	113.78	118.00
3	infinite number of cases				382.38	352.86
4	143.24	63.67	—	—	103.45	155.42
5	297.78	—	—	—	297.78	318.00
6	522.93	558.66	548.53	582.93	553.26	601.21
7	101.00	312.41	458.70	—	296.13	351.05
Total					1845.74	2019.54

to be able to abandon the ContourFollowing behavior (consult section 2.6.2 for a deeper insight into the algorithm *Bug2*).

3.3 The Dynamic Window Approach with no Local Minima

In this section, the *Dynamic Window Approach (DWA)*, briefly discussed in chapter 2, will be extended to properly avoid those trapping situations caused by the *local minima* problem — as was previously done with *PFMs*. Specifically, such an improvement will be accomplished by incorporating our *navigation filter* into the advanced scheme for robot control used by *DWA*. As a final part of this section, the new resulting *DWA*-based strategy will be widely tested in both real and simulated scenarios with the purpose of clearly demonstrating the achievement of the desired local minima-free property. What is more, the aforesaid experimentation will also serve to show the presence of two additional properties, not found in the original *DWA*, which namely are: on the one hand, the ability to solve complex navigation tasks even when the detection of obstacles relies on low-cost sensors; and, on the other hand, the easiness to find a suitable setting of the algorithm's parameters for the specific mission at hand (as

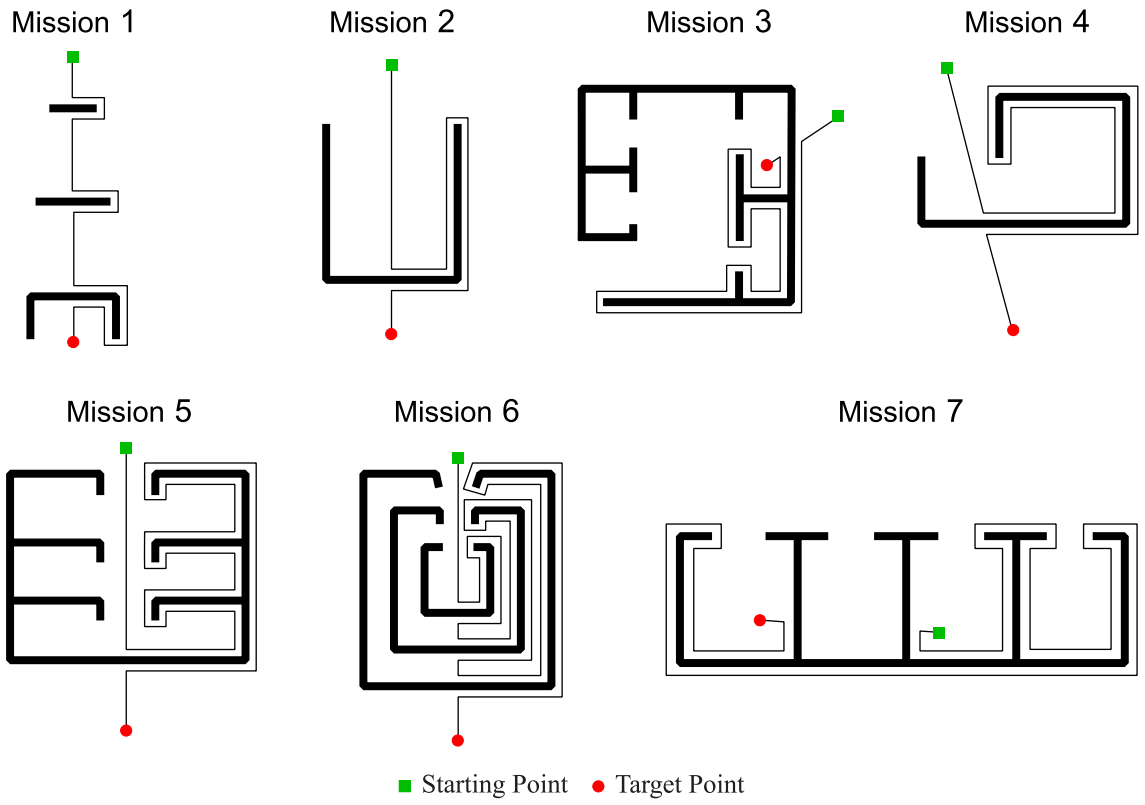


Figure 3.18: Expected results for the algorithm *Bug2* from mission 1 to 7.

Table 3.6: Relative performance of *Random* T^2 and *Bug2*.

Mission	1	2	3	4	5	6	7	Average
$\frac{\text{Bug2}}{\text{Random } T^2}$	1.24	1.04	0.92	1.50	1.07	1.09	1.19	**1.15**

a justification for this last advantage, notice that it essentially comes from the fact that the values given to the configuration parameters of the herein-proposed strategy never —or minimally— compromise / put at risk critical navigation issues such as safe and predictable robot motion).

3.3.1 Going Deeply into the Dynamic Window Approach

Under the name of the *Dynamic Window Approach (DWA)*, there is a very popular obstacle avoidance technique that achieves high-speed and safe navigation of a synchro-drive robot by taking into account its kinematic and dynamic constraints. *DWA* was first introduced in [41] and later extended in [39, 43, 44, 70]. As a general idea, the original *DWA* involves directly searching in the velocity space for the motion command which maximizes a certain objective function.

Entering into further details, the search space —V_p— is given by the set of tuples (v, w) that results from combining all possible translational —v— and rotational —w— velocities of

the robot, as suggested by equation 3.14 (in this equation, v_{max} and w_{max} denote, respectively, the maximum translational and rotational velocities; additionally, observe that v is restricted to exclusively positive values, which means that the robot is never allowed to move backwards). Here, it is important to note that not every tuple in V_p is actually considered during the search performed by *DWA*. More exactly, *DWA* significantly reduces the initial search space V_p by removing from it those tuples that are not classified as both *reachable* and *admissible*. In that regard, a tuple (v, w) is said to be:

- *reachable* if the robot can accomplish such a velocity (v, w) within the next control loop. Or in more formal words, if the tuple (v, w) belongs to the set V_r defined by equation 3.15, where v_c and w_c are the current translational and rotational velocities of the robot, $\dot{v}_{max}$ and $\dot{w}_{max}$ represent the maximum translational and rotational accelerations —or decelerations with a minus sign—, and, at last, Δ_t is the duration of the control loop.

- *admissible* if the robot can come to a complete stop without hitting any obstacle over the trajectory inherently associated with the regarded tuple (v,w). $\Big($As made evident in figure 3.19(a), in synchro drive-type robots, a velocity (v, w) implies a movement on a circular trajectory / arc with constant curvature —to be precise, this curvature is $\frac{w}{v}\Big)$. Rewriting the above in formal terms, the *admissibility* of a tuple (v, w) does demand to be part of the set V_a (see equation 3.16). As can be observed, this set of *admissible* velocities is essentially computed by means of the function $Dist(v, w)$ that evaluates the distance to the nearest obstacle along the circular arc determined by its parameters v and w (for the sake of clarity, figure 3.19(b) shows how the function $Dist$ works).

In short, the search space of *DWA* is characterized by the expression $V_r \cap V_a$ (from now on, such a definitive search space will be referred to as V_d). By way of example, figure 3.19(d) illustrates the calculation of V_d for the situation presented in figure 3.19(c).

$$V_p = \Big\{(v, w) \mid v \in [0, v_{max}],\ w \in [-w_{max}, w_{max}]\Big\}. \tag{3.14}$$

$$V_r = \Big\{(v, w) \mid (v, w) \in V_p,\ \frac{v - v_c}{\Delta t} \in [-\dot{v}_{max}, \dot{v}_{max}],\ \frac{w - w_c}{\Delta t} \in [-\dot{w}_{max}, \dot{w}_{max}]\Big\}. \tag{3.15}$$

$$V_a = \Big\{(v, w) \mid (v, w) \in V_p,\ v \leq \sqrt{2 \cdot Dist(v, w) \cdot \dot{v}_{max}},\ w \leq \sqrt{2 \cdot Dist(v, w) \cdot \dot{w}_{max}}\Big\}. \tag{3.16}$$

As a final step, *DWA* applies a process of optimization which consists in finding the velocity tuple in V_d that provides the highest utility on the basis of an objective function named G. Broadly speaking, this function includes terms that trade-off driving the robot at a fast speed, oriented to the target, and far away from obstacles. Equation 3.17 reveals more details about function G. To this respect, first of all, notice that μ_1, μ_2, and μ_3 are the weighting factors for the three aforementioned terms $\Big($to be more exact, they satisfy the following conditions: $\mu_i > 0\ \forall i = 1, 2, 3$ and $\sum_{i=1}^{3} \mu_i = 1\Big)$. Secondly, with regard to the components of G, the *Speed* function is used to strongly favor high-speed navigation, as can be deduced by examining equation 3.18. On the other hand, the *Align* function measures the angular error between the target direction and a look-ahead estimation of the heading that the robot would

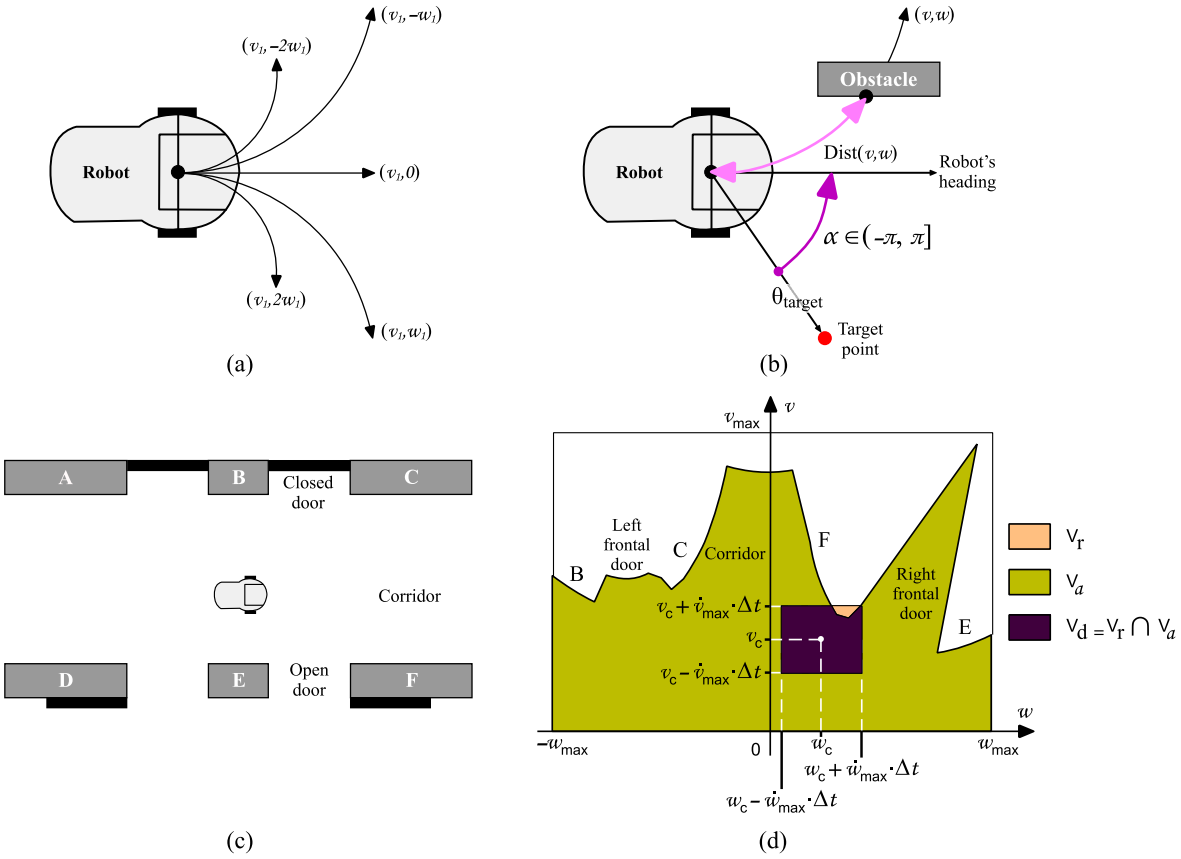


Figure 3.19: More about *DWA*: (a) circular arcs corresponding to different values of v and w; (b) the *Dist* function and the α angle; (c) a particular navigation environment with a certain robot pose; (d) V_r, V_a, and V_d in (c).

have by moving with a constant rotational velocity w during the interval of time of the next control loop. Moreover, to say the obvious, the bigger the angular error, the larger the penalty imposed by the *Align* function on the velocity tuple under consideration (see equation 3.19[10] —and also figure 3.19(b) for a graphical interpretation of the involved angle α). To conclude, concerning the *Dist* function, no explanation is really needed since it has been previously described in the context of the *DWA*'s search space.

After calculating the maximum of the objective function G over V_d, the corresponding tuple becomes the new motion command for the robot. This calculation is repeated every Δt.

$$G(v, w) = \mu_1 \cdot Speed(v) + \mu_2 \cdot Align(w) + \mu_3 \cdot Dist(v, w). \tag{3.17}$$

$$Speed(v) = \frac{v}{v_{max}}. \tag{3.18}$$

$$Align(w) = 1 - \Big(Norm(\alpha - w \cdot \Delta t)\Big). \tag{3.19}$$

3.3.2 The Navigation Filter as a part of the Dynamic Window Approach

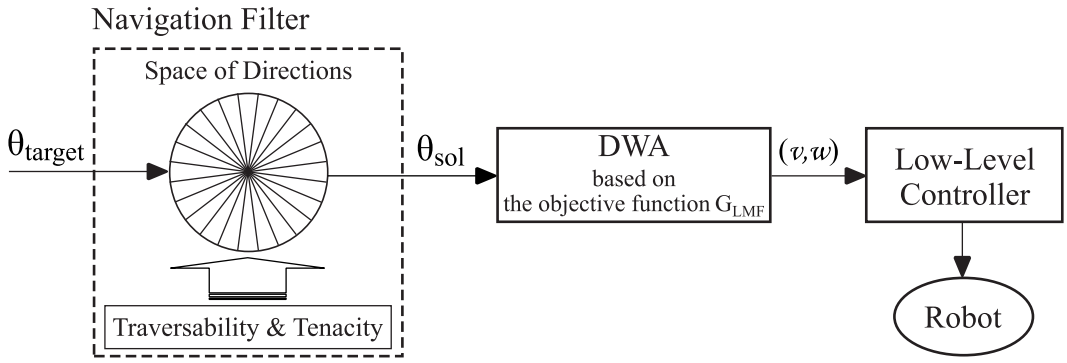
As explained in section 3.1, the navigation filter provides as output θ_{sol}, which is a direction that enables the robot to escape from any local minimum found while pursuing the target configuration. Keeping this in mind, a local minima-free (*LMF*) version of the *Dynamic Window Approach* can be gained by simply establishing θ_{sol} —and not θ_{target}, as originally done in *DWA*— as the preferred direction of motion for the robot. To this end, a change is required on the *Align* component of the objective function that is used to evaluate how good a velocity command / tuple (v, w) actually is. As formally defined by equation 3.20, our suggested align-type function named $Align_{LMF}$ assigns the highest utility to the velocity tuple(s) expecting to produce the perfect alignment of the robot's heading along the θ_{sol} direction (or in other words, the closer a tuple gets from such an alignment, the higher is the value returned by $Align_{LMF}$). Equation 3.21 shows the whole expression for the objective function of the new *DWA*-based strategy being proposed (in G_{LMF}, the *Speed* and *Dist* functions, and the weighting factors μ_i correspond to the ones of the classical *DWA* previously described in section 3.3.1).

To close, figure 3.20(a) presents a basic block scheme of the proposal discussed above, while figure 3.20(b) illustrates the angle α' that measures the current alignment error between the robot and θ_{sol} (notice that this angle is included in the equation for $Align_{LMF}$).

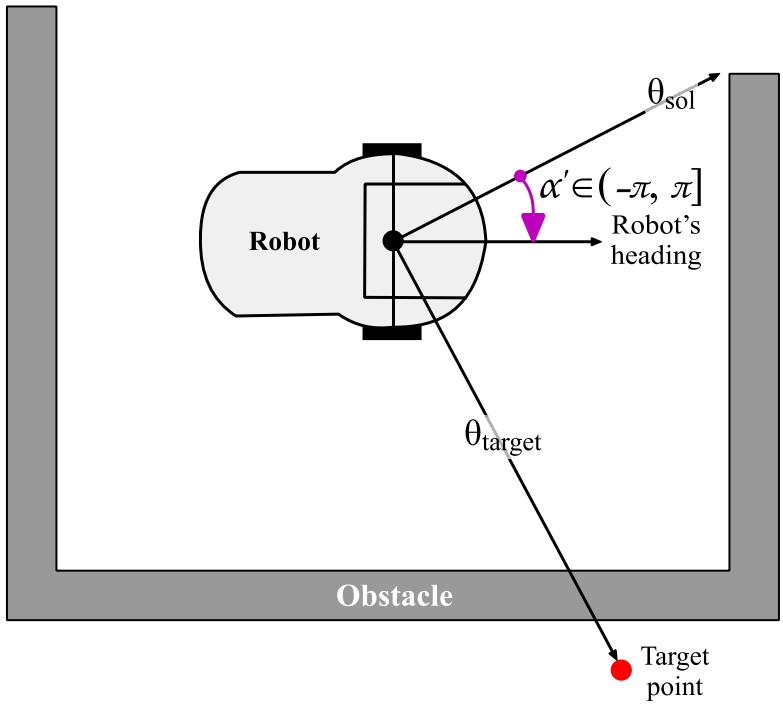
$$Align_{LMF}(w) = 1 - \Big(Norm(\alpha' - w \cdot \Delta t)\Big). \tag{3.20}$$

$$G_{LMF}(v, w) = \mu_1 \cdot Speed(v) + \mu_2 \cdot Align_{LMF}(w) + \mu_3 \cdot Dist(v, w). \tag{3.21}$$

[10] In that equation, the auxiliary *Norm* function computes the normalized value of an angle, which is the equivalent angle in the range $(-\pi, \pi]$



(a)



(b)

Figure 3.20: The use of the navigation filter for building a local minima-free *Dynamic Window Approach*: (a) as a fundamental modification to the original *DWA*, those velocity tuples moving the robot in the direction of θ_{sol} are now favored by the new objective function G_{LMF}; (b) exemplification of the α' angle involved in the calculation of the $Align_{LMF}$ function.

3.3.3 Experimental Evaluation of the Absence of Influence of Local Minima, as well as of an Additional Feature Gained over *DWA*

The aim of this section is two-fold: first of all, to perform a set of both real and simulated experiments that clearly exhibits the absence of the local minima problem when navigating according to the new *DWA*-derivative approach put forward in figure 3.20(a); and, secondly, to demonstrate how easy is the choice of the weighting factors —μ_1, μ_2, and μ_3— that are included in the objective function of such a novel approach (see equation 3.21).

Before proceeding with the above, however, some discussion is needed to clarify a functional aspect of the key component of our proposal. We are referring to the navigation filter, and more exactly, to the criterion that it adopts for choosing the direction —either left or right— to follow the contour of the detected obstacles. Recall that this criterion was left partially open-ended when explaining the navigation filter at the beginning of the chapter. Further in that respect, in section 3.1.4, three alternative criteria were established to automatically select the aforesaid contour following direction. Specifically, they were described under the names of *minimum-turn* criterion, *random* criterion, and *fixed-beforehand* criterion. Getting to the point, in all the forthcoming tests, it is assumed that the navigation filter uses a mixture of the first- and last-listed criteria to make the type of decision being considered. In short, this assumption means that, in the event of the finding of a new blocking obstacle, the navigation filter forces the robot to follow its contour in the same direction as the one taken for the previously detected obstacle. Besides, in the special case of the detection of the first obstacle[11], the navigation filter chooses the left or right contour following direction according to the minimum turning angle that locally aligns the robot's heading with the obstacle boundary. As a final comment, notice that the term *Unvarying* T^2 will hereafter be employed to designate the purely reactive strategy of figure 3.20(a) in the particular context of deciding about the contour following direction as it has just been stated.

3.3.3.1 Tests with Real Robots

The strategy *Unvarying* T^2 was tested in three scenarios containing some of the obstacle configurations that typically lead to the trapping of the robot —always under the context of purely reactive navigation. From such a set of experiments, the first two were conducted using a Pioneer 3-DX robot, while the last one involved the miniature robot Soccerbot (refer to appendix A for a detailed description of these robots).

Figures 3.21(a) and (b) show the characteristics of the environment where the first test was carried out. As can be seen, the scenario essentially consisted of many small obstacles partially enclosing the target point. Furthermore, this point was only reachable through an entrance situated in the opposite side of the initial robot location. The resultant trajectory of about 18 meters long is plotted in figure 3.21(a).

In the second test, various cardboard-type boxes were employed to build two canyons —one U-shaped and the other L-shaped— and a wall, which were distributed in the environment in a way that the robot was forced to overcome, one by one, these three main obstacles to attain the desired target $\Big($look at figures 3.21(c) and (d) for a whole understanding of the mission set up$\Big)$. In this challenging scenario, the *Unvarying* T^2-based robot did navigate along the path that is depicted in figure 3.21(c) (notice that, on this occasion, such a successful path was 28 meters in length).

[11] This case is special because there is not a previous obstacle on which to base the decision whether to move the robot in the left or right contour following direction

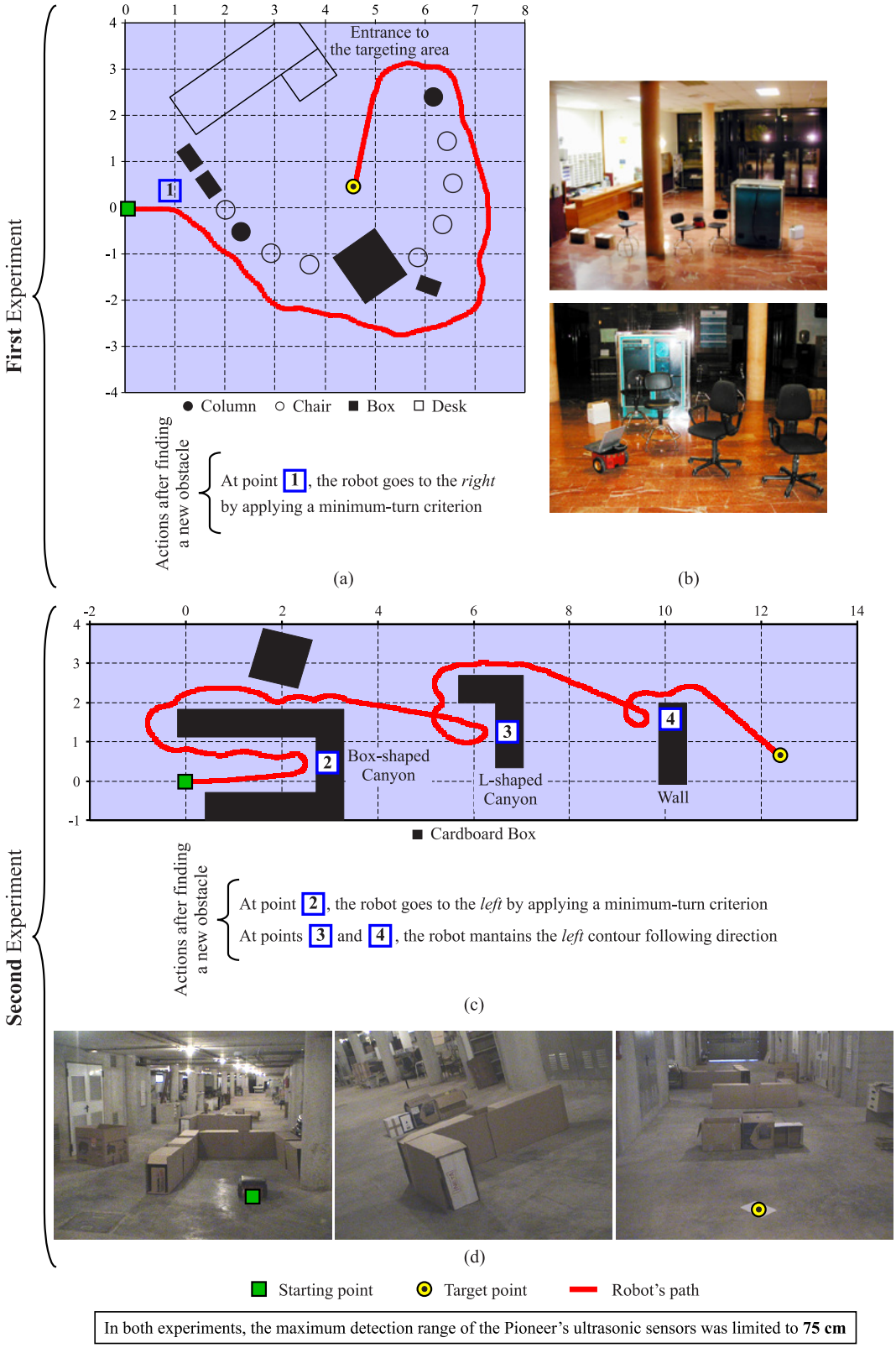


Figure 3.21: Testing the strategy *Unvarying* T^2 on a Pioneer-type robot in two scenarios of increasing complexity. Regarding the first / second experiment: (a) / (c) resultant trajectory reconstructed with the odometry data; (b) / (d) several views of the navigation environment. As an extra material, a ≪video≫ of the second experiment can be found in the electronic version of the document.

The third —and last— test was performed in an office-like environment, as it is illustrated in figure 3.22. To be precise, the mission consisted in going out from one end of the office to the central corridor. With the aim of further increasing the difficulty of the navigation task, several obstacles were artificially created by means of planks, such as, for instance, a box-shaped canyon. What is more, the corridor was crowded with people, meaning that there was a high possibility that the robot might find some of such dynamic obstacles when trying to achieve the target position. As for the results obtained, the mission was successfully completed by the robot by following the 9-meter path represented in figure 3.22(a). As can be observed, the robot, after getting to the corridor, had to avoid an unexpected walking person which was impeding its final progress towards the target.

3.3.3.2 Tests under Simulation

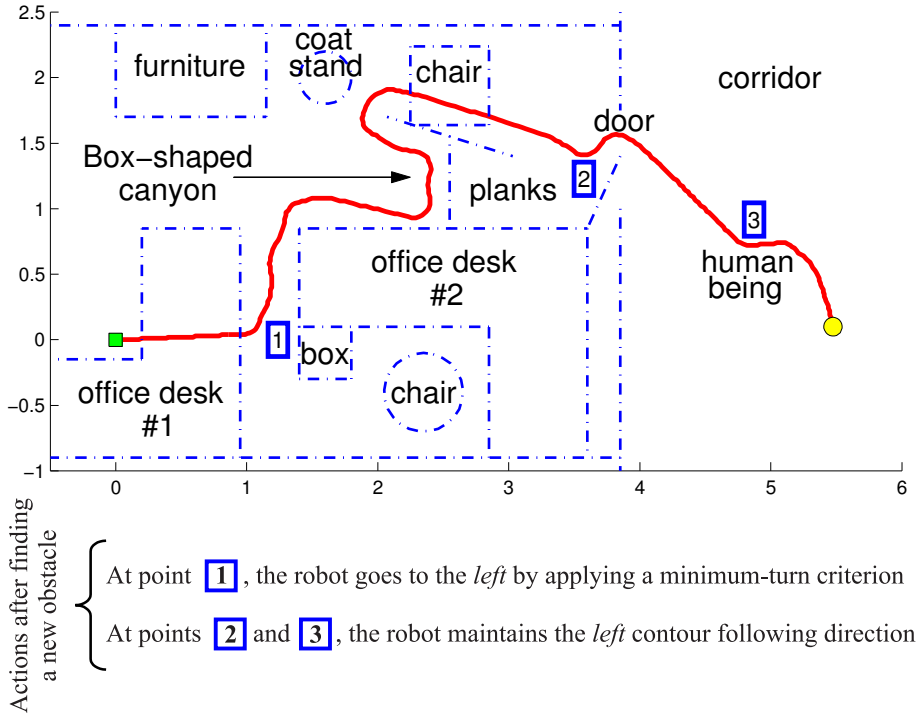
After having just demonstrated the feasibility of the strategy *Unvarying* T^2 for being both implemented and executed on a real robotic platform, we focus now on evidencing, through simulations, two key aspects —one functional and the other related to configuration issues— of such a new purely reactive navigation method. In a few words, the first aspect to be examined is the effective avoidance of local minima in complex scenarios, while the second one refers to the lack of effort that is generally required to choose appropriate values for the internal parameters of *Unvarying* T^2.

In the following, the testing for the above aspects of the strategy *Unvarying* T^2 is performed by using the so-called *MobileSim* simulator, which is a free software developed by *Adept MobileRobots*. More to the point, among the various models of robots that MobileSim currently supports, a *Pioneer 3-DX* vehicle is simulated in all the forthcoming tests —see appendix A for a detailed description of this specific two-wheel drive robot.

Navigation among Troublesome Obstacles

A series of experiments was carried out so as to exhibit the built-in ability that a robot navigating in accordance with the strategy *Unvarying* T^2 has to escape from intricate obstacle configurations. As can be observed in figures 3.23 and 3.24, eight different environments were considered for experimentation (as a way of confirming the difficulty of these environments, notice that, in most of them, the algorithm *DWA* would certainly lead the robot to an undesirable trapping situation). Such environments are similar —moreover, some of them identical— to those which were used to test the previously proposed *PFM*-derivative strategy named *Random* T^2 (see figures 3.16 and 3.17 for comparison).

Moving on the results, the *Unvarying* T^2-based robot was able to successfully reach the given target in each of the above-mentioned environments (figures 3.23 and 3.24 show, in red, the set of trajectories followed by the robot). Regarding these results, it is important to stress that, to complete some of the experiments / missions, the robot had to overcome several local minima —for instance, three in Mission 5—, or to circumnavigate a spiral-like obstacle from its most inner loop —Mission 6—, or even to find the exit of a maze —Mission 7. In addition to this potential for solving complex navigation tasks, it should also be said that, across all missions, the average linear velocity of the robot was 78 cm / s, and maximum velocities of up to 1 m / s were achieved. This feature of the strategy *Unvarying* T^2 for driving the robot at a high speed is directly inherited from *DWA*.



(a)

(b)

Starting point Target point Robot's trajectory

The maximum detection range of the Soccerbot's infrared sensors was limited to **40 cm**

Figure 3.22: Testing the strategy *Unvarying* T^2 on the low-cost robot Soccerbot: (a) resultant path reconstructed with the odometry data; (b) partial views of the office-type environment. Besides, a ≪video≫ of this experiment can be found in the electronic version of the document.

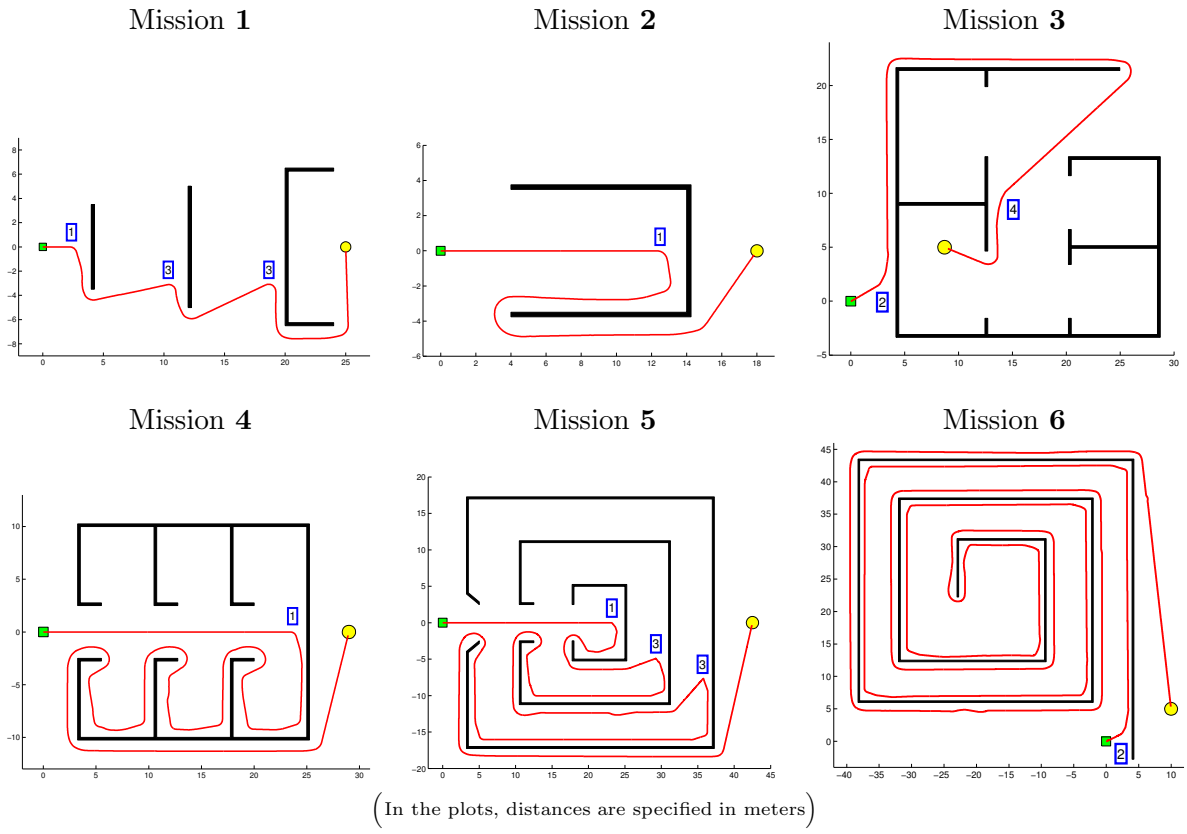


Figure 3.23: Testing the strategy *Unvarying* T^2 over scenarios where purely reactive robots get typically trapped in local minima.

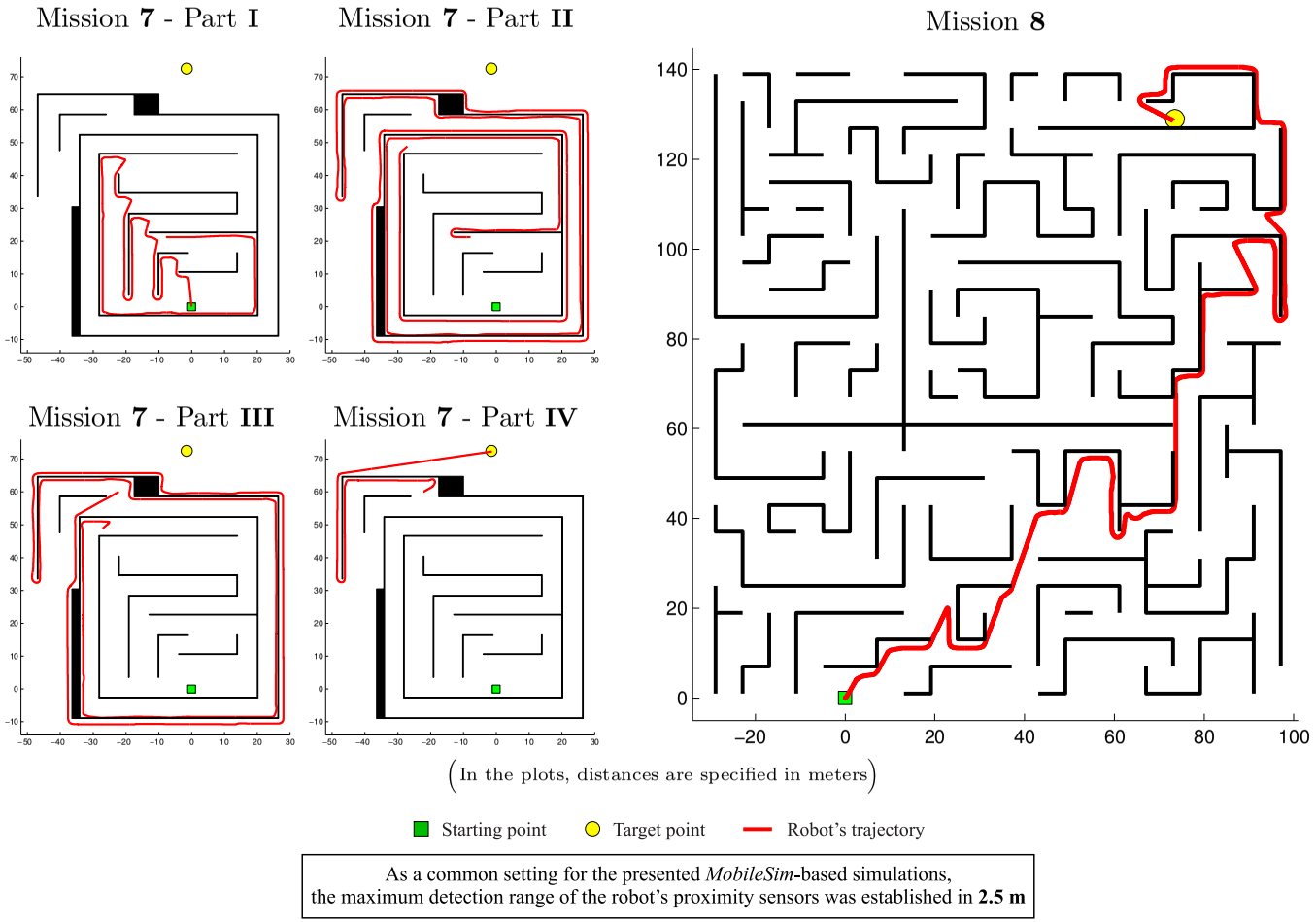


Figure 3.24: Testing the strategy *Unvarying* T^2 over two large-scale scenarios having obstacles placed in a maze-like form. As can be seen, multiple plots are given for Mission 7 with the intention of showing the resultant robot's path with no overlapping. Additionally, notice that, in the above two missions, the particular contour following direction taken by *Unvarying* T^2 after the detection of each new obstacle is not highlighted as it was in figure 3.23, mainly due to, on this occasion, such information may make difficult to understand how the trajectory of the robot goes around the environment obstacles.

Greatly Simplifying the Task of Parameter Tuning

The *Dynamic Window Approach* and our novel variant *Unvarying* T^2 are two strategies which require the setting of exactly the same parameters. Specifically, they choose the next velocity command for the robot based on the use of an objective function that is defined as a weighted sum of three terms, namely *Speed*, *Align* / $Align_{LMF}$, and *Dist* (see equations 3.17 and 3.21). Accordingly, it seems clear that there is a weight factor associated with each of the aforementioned terms. From a practical point of view, these weights, generically denoted as μ_i with $i \in \{1, 2, 3\}$, determine the trade-off that the corresponding strategy makes among navigation speed, θ_{target}-directness / θ_{sol}-directness, and robot safety. In this respect, in the following, we are intended to find out how difficult is the achievement of a suitable trade-off by means of the modification of the μ-weights in both *DWA* and *Unvarying* T^2; or in other words, we focus on revealing how much influence the μ-weight values do actually have on the resultant trajectory generated by the *DWA* and *Unvarying* T^2 algorithms.

In keeping with the above purpose, a test was conducted by simulation with *DWA* and *Unvarying* T^2 for each possible value of the μ-weights. The particular environment that was set up for this testing is depicted in figure 3.23 under the name of *Mission 1* (notice that such an environment permits evaluating the effectiveness of the aforesaid strategies in circumnavigating long obstacles / walls). Regarding the μ-weights —μ_1, μ_2, and μ_3—, their set of potential values was reduced by advancing in steps of 0.1 over the real range (0,1]; that is to say, we assumed that $\mu_i \in \{0.1, 0.2, \ldots, 1.0\}\ \forall i = 1, 2, 3$. Besides, these weights were forced to satisfy the equality $\sum_{i=1}^{3} \mu_i = 1$. Getting to the point, the *DWA* and *Unvarying* T^2 algorithms were executed in mission 1 using a total of 36 different values of the μ-weights.

Figures 3.25(a) and (b) present in a single plot the 36 mission 1-solution paths which resulted from configuring the μ-weights of, respectively, *DWA* and *Unvarying* T^2 as suggested above. Additionally, table 3.7 shows the best and worst values as well as the average and the standard deviation (σ) of three important attributes of the paths that appear in each plot of figure 3.25. In short, the *length* attribute, by simply observing σ, gives us a clear idea about the length variability of the paths. The *time-to-completion* attribute, on the other hand, allows us to know if the robot got temporarily stuck in certain areas of the environment for some settings of the μ-weights. Lastly, how safe were the paths can be determined by the remaining attribute named *virtual collisions*. This attribute refers to the number of times that the robot moved inside the so-called safety area of an obstacle while navigating to the target (as shared by *DWA* and *Unvarying* T^2, the locally sensed obstacles were mapped in the configuration space, or C-space, by growing them by the robot's shape plus a safety area; consequently, the location of the robot in that area plainly means to be very close to an obstacle, or equivalently, to be in the event of a significant risk of collision).

By observing / deeply analyzing the contents of figure 3.25 and table 3.7, it seems obvious that the strategy *Unvarying* T^2 is largely insensitive to the value of the μ-weights. Under the change of these values, our strategy provides almost the same solution in terms of path length, navigation speed, and safety from obstacles. Furthermore, this solution broadly coincides with the one found by *DWA* when using its best setting of the μ-weight parameters. In essence, all the above means that the tuning of parameters is not really a necessary task for *Unvarying* T^2, since this strategy does inherently achieve a good trade-off among path length performance, robot's speed, and robot's safety, quite regardless of the μ-weights. Finally, it is important to note that this simplicity of the parameter tuning is not shared with *DWA* (as experimentally evidenced in figure 3.25(a), the *DWA* algorithm may generate widely dif-

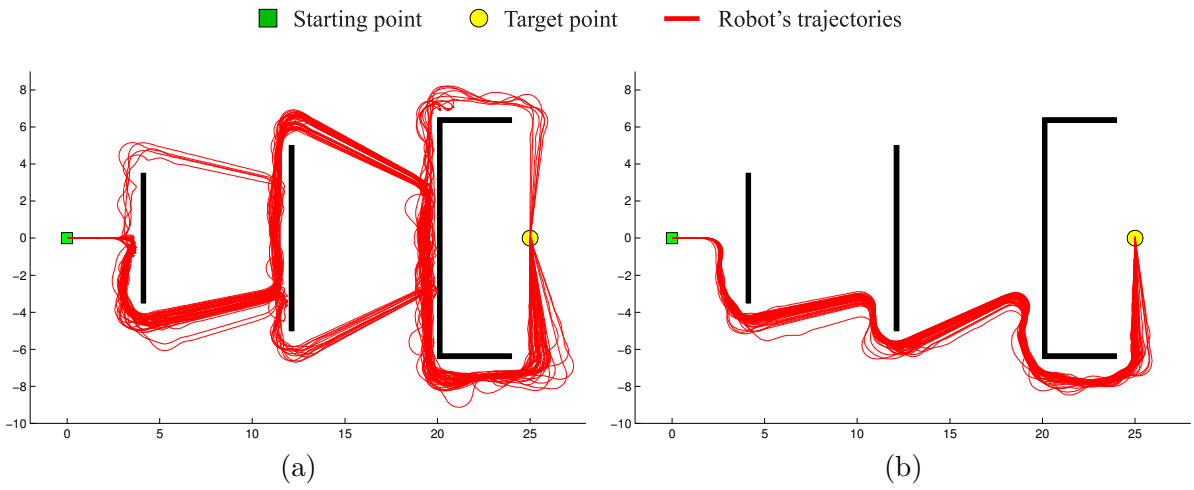


Figure 3.25: Visual assessment of the influence of the value of the μ-weights on the trajectory produced by (a) *DWA* and (b) *Unvarying* T^2.

Table 3.7: Quantitative assessment of the influence of the value of the μ-weights on the trajectory produced by *DWA* and *Unvarying* T^2.

		Length (m)	Time to Completion (number of robot cycles)	Virtual Collisions (number of risky situations)
Algorithm *DWA*	Best (shortest|smallest|smallest)	46.06	996	8
	Worst (longest|largest|largest)	88.02	2981	990
	Average	57.58	1329	167
	Standard Deviation	6.07	450	270
Algorithm *Unvarying* T^2	Best (shortest|smallest|smallest)	40.14	719	1
	Worst (longest|largest|largest)	42.39	793	15
	Average	40.99	745	6
	Standard Deviation	0.59	17	3

ferent paths by varying the μ-weight values, so that one should be very careful in the setting of these parameters in order to obtain good navigation results).

3.4 Pros and Cons

There are several significant advantages (A), and also some disadvantage (D), derived from doing navigation on the basis of the new concepts of *Traversability and Tenacity* $\left(T^2\right)$. They all are briefly discussed next.

A1. T^2 are two general concepts which can be directly applied on a large variety of currently existing purely reactive navigation methods for definitely solving the well-documented problem of *local minima*. Or in other words, a T^2-based method does acquire the skill for successfully moving the robot out of a local minimum regardless of: (1) the precise shape and size of the obstacle/s that is/are causing such a local minimum; and (2) the maximum obstacle detection range of the robot's sensors (as a way of exemplifying the non-influence of points 1 and 2, notice that a robot controlled by T^2 would be able to escape from an extremely deep and wide U-shaped canyon by merely relying on contact/zero-range sensors).

A2. T^2 keeps the *purely reactive* essence of the method on which it is applied. Consequently, the high responsiveness of a T^2-based method makes it well-suited for navigating in both unknown and dynamic environments.

A3. T^2 provides a purely reactive method with the capacity for being used on *low-cost* robots, typically equipped with sensors that offer noisy perceptions of the environment as well as just a partial coverage of the local robot's surroundings.

D1. T^2 may fail to detect narrow passages / gaps between obstacles where the robot actually fits in. For some scenarios —such as the one of figure 3.26—, this gap-detection failure entails the loss of opportunities to reach the target through a shorter path.

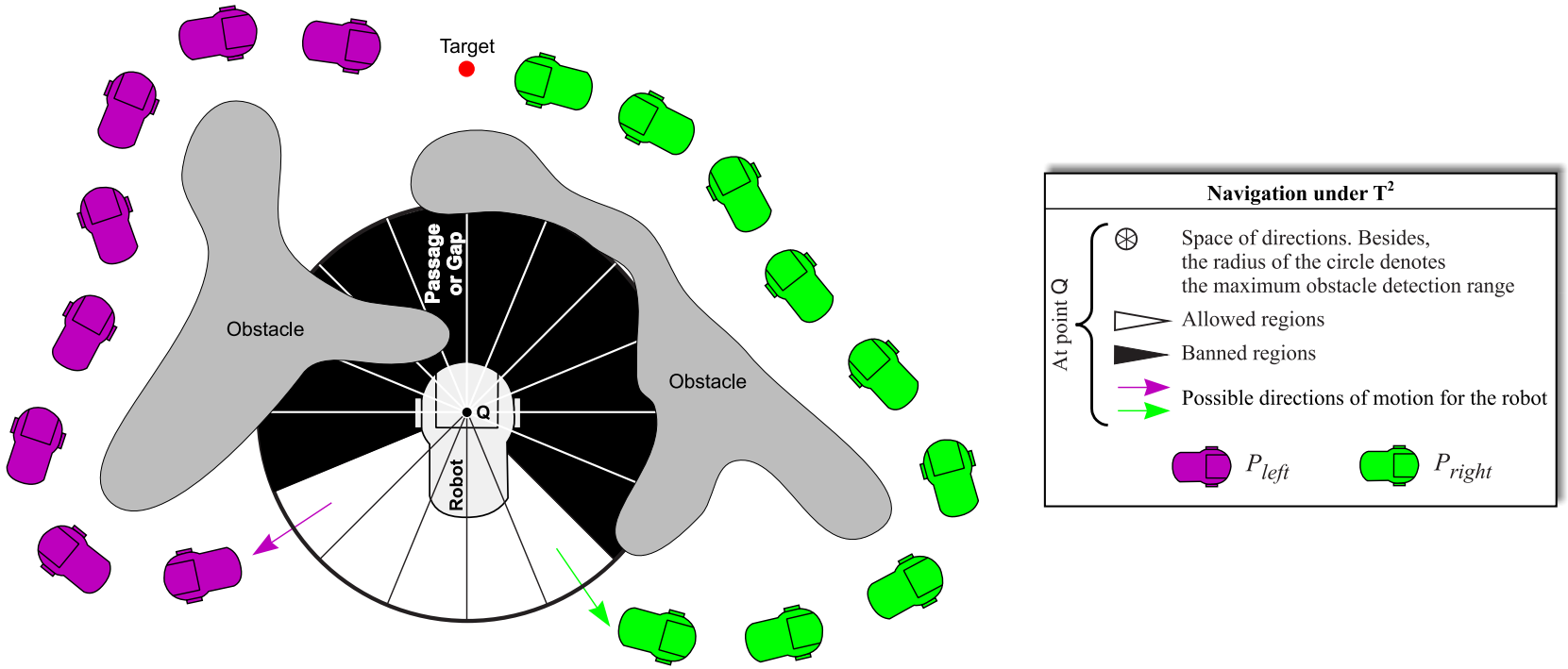


Figure 3.26: A drawback in the application of the concepts of *Traversability and Tenacity*. As can be observed, none of the two possible motion directions generated by T^2 at point Q —these directions are marked with colored arrows— move the robot towards the middle passage, which represents the shortest way of getting to the target point. As a general remark, it is important to say that T^2 inherently favors the navigation of the robot across the open spaces of the environment, where naturally there is less risk of collision. According to this, in the above scenario, the whole path planned by T^2 would be one of those labeled as P_{left} and P_{right}.

Achieving a Better Path Length Performance for the Algorithm *Bug2*

This chapter focuses on the popular family of reactive / sensor-based algorithms named *Bug*, which was partially covered in section 2.6. Informally speaking, the members of this family mainly stand out for imitating the way in which some insects reach a certain —global— target as moving through completely unknown environments. To be fair, these algorithms go beyond this imitation by guaranteeing, from a theoretical point of view, the successful completion of the navigation task whenever possible. Moreover, such a completeness property is achieved by essentially using local knowledge coming from the immediate robot surroundings. Lastly, notice that, precisely due to this local character, Bug-like algorithms tend to be very efficient in terms of both processing time and memory consumption.

Through the next pages, we put forward an enhanced version of one of the members of the family Bug with best balance between simplicity and performance, according to the recent study presented in [71]. To be exact, we are referring to the classical algorithm *Bug2*, which was described in [53] by Lumelsky and Stepanov. The new version of this algorithm, named *Bug2+*, preserves the simplicity of the original approach while improving its path length performance.

4.1 Related Work: The Algorithm *Bug2*

In the following, the fundamentals of the algorithm *Bug2* are examined in depth.

4.1.1 Assumptions

The algorithm Bug2 makes several assumptions about the robot and the environment, which are highlighted next:

- The mobile robot is considered to be a point equipped with a complete set of error-free tactile sensors. This point-shaped robot is capable of moving everywhere in free space as well as along the contour of obstacles. Finally, the problem of localization is supposed to be solved so that the vehicle knows its current position and the one of the target.
- As for the navigation environment, it is assumed to be static, unknown, and two-dimensional. In addition, the admissible shapes for the obstacles are conditioned by the *Jordan curve theorem* [72], which forces the contour of each obstacle to define a simple[1] and closed curve of finite length. Besides, obstacles should not touch each other.

[1] A curve is simple if it does not cross itself

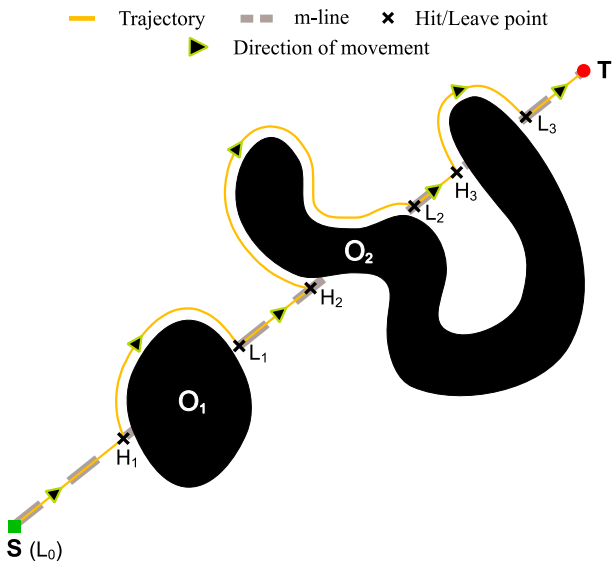


Figure 4.1: A path planned by the algorithm Bug2. The direction for following the contour of the obstacles was supposed to be *left* ($pCFD$ = left).

4.1.2 Notation

$S, T \in \mathbb{R}^2$ are, respectively, the starting and the target points of the mission. XY ($X, Y \in \mathbb{R}^2$ and $X \neq Y$) represents the straight-line segment with endpoints X and Y. The line connecting the starting and the target points, ST, is referred to as *main line*, or *m-line* in short. On the other hand, O_i denotes a certain obstacle of the environment and ∂O_i its contour curve. Finally, $d(X, Y)$ is a function which measures the Euclidean distance between any two points X and Y ($X, Y \in \mathbb{R}^2$).

4.1.3 Description

The algorithm Bug2 exhibits two different behaviors: motion-to-goal and boundary-following. During the former, which is activated first, the robot moves towards the target (T) along the m-line. The boundary-following behavior, on the other hand, is invoked when the robot encounters an obstacle (O_i) on its way. The point where this obstacle is found is called *hit point* (H_j). Next, the robot follows the contour of the obstacle (∂O_i) to the left or right according to a user-definable parameter named $pCFD$. During this contour-following process, it may so happen that the robot returns to H_j meaning that a loop around the obstacle boundary has been completed. In such a case, the target is inside the obstacle, not being thus achievable. More usual is, however, the situation where the robot gets to a point on the m-line closer to T than H_j. At that moment, a *leave point* (L_j) is defined and the motion-to-goal behavior is invoked again. The motion-to-goal and boundary-following behaviors alternate the control of the robot in the way explained above until either T is reached or the planner becomes aware of the impossibility of finding a solution to the problem. Algorithm 4.1 provides a deeper description of the strategy Bug2, and figure 4.1 shows its execution in a simple scenario.

The length of a path generated by Bug2 never exceeds the limit given by expression 4.1, where i denotes an obstacle of the scene (O_i), n_i represents the number of intersections of ∂O_i with the m-line, and B_i refers to the O_i's perimeter. Such an upper bound can be significantly improved as stated by expression 4.2 when assuming that obstacles are convex.

Algorithm 4.1 The algorithm *Bug2* step by step

Params:	$pCFD \in \{left, right\}$
Step 0:	Initializations
Step 1:	Motion-to-goal behavior
Step 2:	Boundary-following behavior

0) Set $j = 1$ and $L_0 = S$

1) Move along the straight-line segment $L_{j-1}T$ until one of the following occurs:

   a) T is reached. The algorithm stops

   b) An obstacle O_i is found. Define the hit point H_j and go to step 2

2) Follow the contour of the obstacle (∂O_i) to the left or right according to $pCFD$ until one of the next three possible situations arises:

   a) T is reached. The algorithm stops

   b) The robot returns to H_j. The algorithm stops because the target is unreachable

   c) The robot gets to a point Q satisfying condition C_1. As a result, either action A_1 or action A_2 is taken depending on whether condition C_2 is met (A_1) or not (A_2)

   C_1: Q is a point on the m-line ($Q \in ST$) such that $d(Q,T) < d(H_j,T)$
   C_2: the straight-line segment QT does not cross the obstacle O_i at point $Q^\star$
   A_1: define the leave point $L_j = Q$, set $j = j + 1$ and, lastly, go to step 1
   A_2: continue in step 2

   * The straight-line segment QT is considered to cross the obstacle O_i at point Q when a segment of QT lies inside O_i in the vicinity of Q

$$d(S,T) + \sum_i \frac{n_i}{2} B_i \tag{4.1}$$

$$d(S,T) + \sum_i B_i \tag{4.2}$$

4.2 The New Algorithm *Bug2+*

In this section, a description of an enhanced version of the algorithm Bug2 called *Bug2+* is provided. The following features of this new Bug-derivative strategy are emphasized:

- Bug2+ ensures convergence to the given target if it is reachable.
- The length of a path planned by the algorithm Bug2+ is always less or equal to the one by Bug2.

- As a direct consequence of the preceding fact, the length of a path produced by Bug2+ never goes beyond the limit defined by expression 4.1.

In addition, it is important to note that the strategy Bug2+ preserves both the simplicity and the intuitive behavioral description by which the algorithm Bug2 is mostly characterized. Next, we focus on the key differences between Bug2 and Bug2+. Besides, regarding the latter approach, a proof is given for the three properties listed earlier.

4.2.1 Changes with respect to the Strategy Bug2

The strategy Bug2+ makes the same assumptions about the robot and the environment as the algorithm Bug2. Moreover, Bug2+ relies on the same two behaviors as Bug2: motion-to-goal and boundary-following.

The planner Bug2+ differs from Bug2 in adopting a special leaving condition, or in other words, in applying a sharper criterion for deciding to invoke the motion-to-goal behavior when the robot is circumnavigating the contour of an obstacle. Remember that, in Bug2, this transition occurs when a point Q on the m-line nearer the target than H_j is found. Furthermore, for really abandoning the boundary-following behavior, the point Q should satisfy condition C_2 as well, which requires the robot to be able to move along the straight-line segment QT without immediately hitting the current obstacle (look at figure 4.2 for a better understanding of condition C_2). The strategy Bug2+, as opposed to Bug2, takes into account the points which do not meet condition C_2 in the decision associated with leaving the contour following process (notice that, in Bug2+ terminology, condition C_2 renames to condition C_4, so that hereafter C_4, and not C_2, will be used for referring to this condition). Let Γ denote the set of m-line's points not fulfilling condition C_4 which have been found by the robot during the last —and still in-progress— activation of the boundary-following behavior. Then, Bug2+ will perform a transition to the motion-to-goal behavior when reaching a point Q on ST with $Q \notin \Gamma$ and satisfying the inequality $d(Q,T) < min\{d(\gamma,T)\ \forall\gamma \in \Gamma\}$ (observe that $H_j \in \Gamma$ according to the definition of *hit point*).

A complete description in pseudocode of the strategy Bug2+ is provided in algorithm 4.2 (those changes with respect to algorithm 4.1 are marked in **bold**). Additionally, figures 4.3(a) and (b) compare the trajectories generated by, respectively, Bug2 and Bug2+ in a scenario consisting of a G-shaped obstacle. As can be seen, the path length of our proposal was significantly better in this experiment, being, approximately, 1.35 times shorter.

4.2.2 Formal Verification of the Bug2+'s Properties

The following proves that: (1) Bug2+ converges to the target when reachable, and (2) Bug2+ is never worse and sometimes better than Bug2 in terms of path length performance.

Lemma 1. *Algorithm 4.2 behaves just like algorithm 4.1 if action A_4 is not taken.*

Assumption/s. Both algorithms are applied to the same scenario with the parameter $pCFD$ set to the same value.

Proof. Algorithms 4.1 and 4.2 only differ in some details regarding the causes which determine the transition from the boundary-following behavior to the motion-to-goal behavior. This proof is precisely focussed on this aspect of the algorithms by demonstrating that such a transition will simultaneously take place at the same point Q in Bug2 and Bug2+ under the assumptions described above.

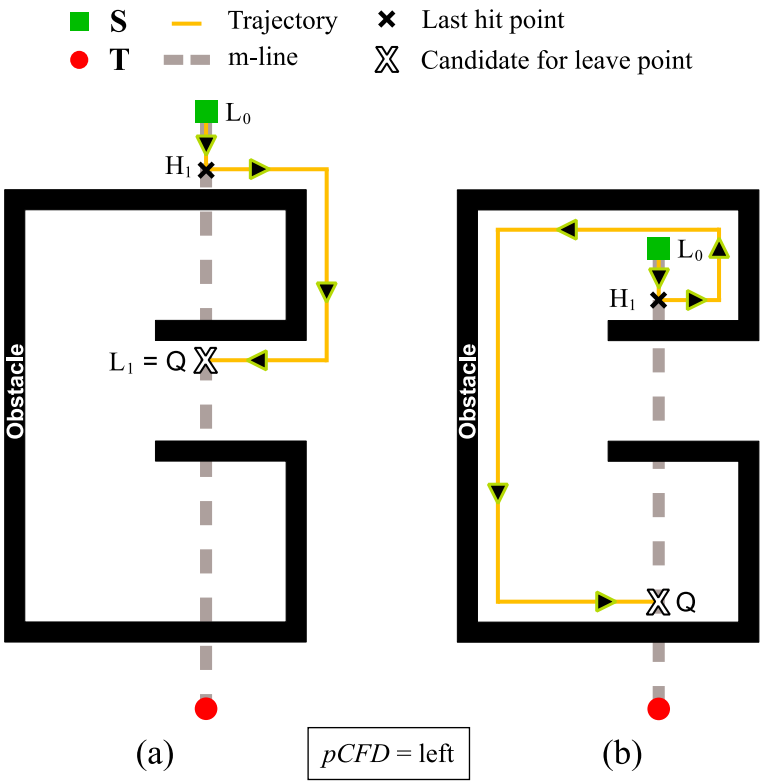


Figure 4.2: Conditions imposed in Bug2 for a transition from the boundary-following behavior to the motion-to-goal behavior: two different situations are considered where condition C_1 is met, i.e. where the robot, while following the contour of an obstacle, reaches the m-line in a point Q which is closer to T than H_j $(j = 1)$. In (b), contrary to (a), the point Q does not become a leave point because the robot cannot progress towards the target by moving along QT (or, in short, because condition C_2 does not hold).

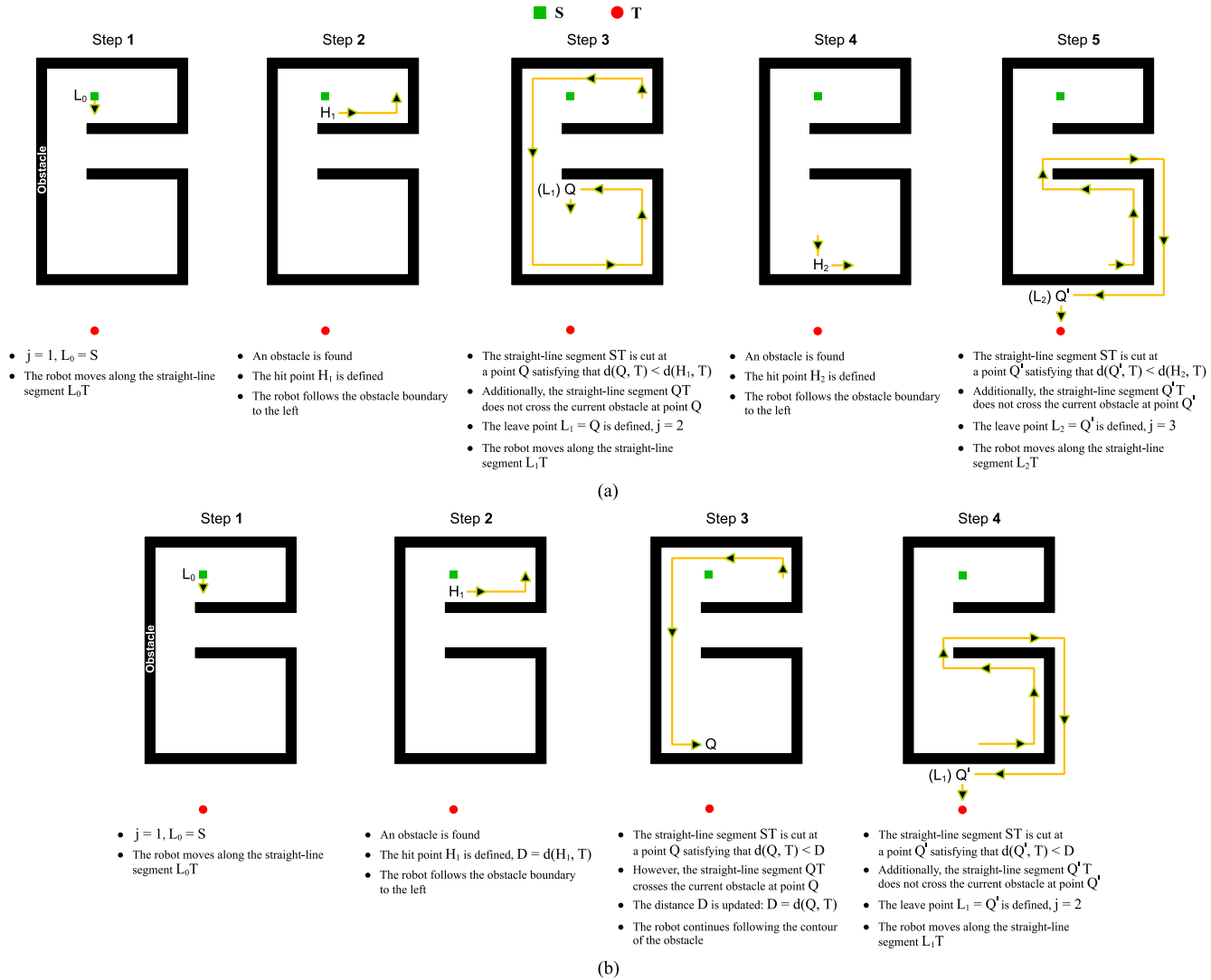


Figure 4.3: Comparing the path length performance of algorithms Bug2 (a) and Bug2+ (b). In both executions, the parameter *pCFD* was set to *left*.

Algorithm 4.2 *Bug2+*: An improvement of the strategy Bug2

Params:	$pCFD \in \{left, right\}$
Step 0:	Initializations
Step 1:	Motion-to-goal behavior
Step 2:	Boundary-following behavior

0) Set $j = 1$ and $L_0 = S$

1) Move along the straight-line segment $L_{j-1}T$ until one of the following occurs:

   a) T is reached. The algorithm stops

   b) An obstacle O_i is found. Define the hit point H_j, **set $\boldsymbol{D = d(H_j, T)}$** and, finally, go to step 2

2) Follow the contour of the obstacle (∂O_i) to the left or right according to $pCFD$ until one of the next three possible situations arises:

   a) T is reached. The algorithm stops

   b) The robot returns to H_j. The algorithm stops because the target is unreachable

   c) The robot gets to a point Q satisfying condition C_3. As a result, either action A_3 or action A_4 is taken depending on whether condition C_4 is met (A_3) or not (A_4)

   C_3: Q is a point on the m-line ($Q \in ST$) such that $\boldsymbol{d(Q, T) < D}$
   C_4: the straight-line segment QT does not cross the obstacle O_i at point Q
   A_3: define the leave point $L_j = Q$, set $j = j + 1$ and, lastly, go to step 1
   A_4: **update $\boldsymbol{D}$ ($\boldsymbol{= d(Q, T)}$)** and continue in step 2

Let us suppose next a situation where both strategies are executing step 2 after defining an identical hit point H_j. In such a case, the ongoing contour following process will be replaced by the motion-to-goal behavior when the robot moves to a point Q satisfying conditions C_1 and C_2 in algorithm 4.1, and conditions C_3 and C_4 in algorithm 4.2. As can be observed, there are no differences between conditions C_2 and C_4. On the other hand, knowing that action A_4 is never taken, the value for the distance D used in condition C_3 will not change from the one given in step 1.b), being thus equal to $d(H_j, T)$. Consequently, conditions C_1 and C_3 are also equivalent, which means that algorithms 4.1 and 4.2 will decide to abandon the boundary-following behavior based on the same criterion, which proves the lemma. □

Definition 1. Let us introduce the concepts of *potential hit point* (H^*) and *potential leave point* (L^*). The former denotes a point where a transition from the motion-to-goal behavior to the boundary-following behavior may occur or, in other words, where the robot may find an obstacle while moving along the m-line. On the other hand, a potential leave point is just the opposite, i.e. a point where a transition from the boundary-following behavior to the motion-to-goal behavior may happen.

Both H^* and L^* points are located on the m-line and, more precisely, in those positions where such a line intersects with the contour of the obstacles as illustrated in figure 4.4(a)

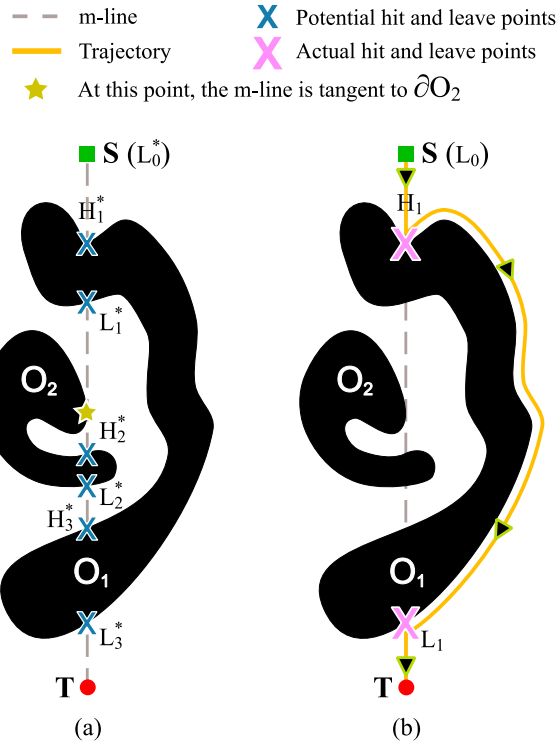


Figure 4.4: Distinguishing between (a) *potential* hit (H^*) and leave (L^*) points, and (b) *real* hit (H) and leave (L) points. In (b), the trajectory of the robot was obtained by applying algorithm 4.2 with the parameter $pCFD$ = left.

(notice that a special potential leave point —L_0^*— is further added at S). This figure also shows that neither H^* nor L^* points are defined when the m-line tangentially intersects with the contour of an obstacle. Remember that algorithm 4.2, like algorithm 4.1, assumes that the robot is able to navigate over the obstacle boundaries and, consequently, through any point of tangency between the m-line and the contour curve of an obstacle. Consider, now, the situation where the robot reaches one of such tangent points by moving along the m-line towards T. At that moment, it seems obvious there is no need for invoking the boundary-following behavior and, therefore, for defining a hit point —and the subsequent leave point— since the robot can continue its straight-line walk to the target. Because of that, no point can be defined as both H^* and L^*. Moreover, taking into account that all H^* and L^* points appear in pairs (H_k^*, L_k^*) for values of k greater than zero —$k \in \mathbb{Z}^+$—, the next inequality holds: $d(H_k^*, T) > d(L_k^*, T)$. Observe, finally, that the points in each pair (H_k^*, L_k^*) are located on the contour curve of the same obstacle.

To conclude, figure 4.4(b) makes evident that just a subset of all H^* and L^* points turns into real hit (H) and leave (L) points when executing algorithm 4.2.

Lemma 2. *In algorithm 4.2, after performing action A_4, the robot will find a point Q satisfying conditions C_3 and C_4.*

Assumption/s. The target point (T) is reachable from S, and T is not located on the boundary of the obstacle being followed when action A_4 is taken.

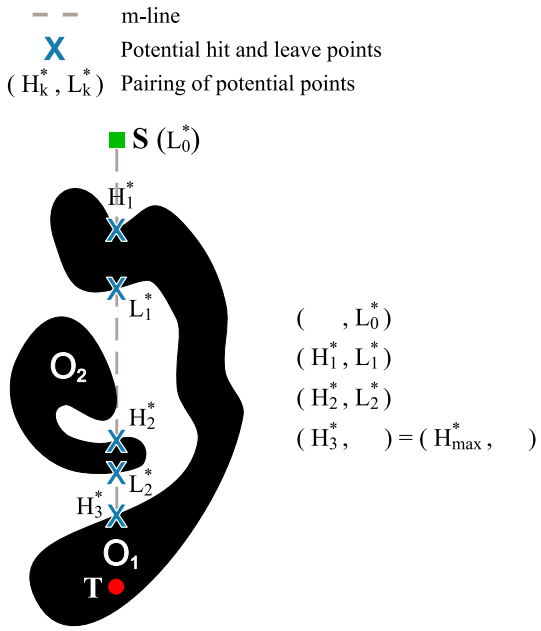


Figure 4.5: An additional unpaired potential point when the target is not accessible.

Proof. Action A_4 is taken when the robot, while following the contour of a certain obstacle —referred to as O_i from now on—, finds an m-line's point Q which fulfills $d(Q,T) < D$ as well as the straight-line segment QT crosses the obstacle O_i at point Q. Such a condition, nevertheless, can be alternatively described by using the concept of potential hit point. Specifically, we can equivalently say that the execution of action A_4 is caused by reaching a potential hit point H_k^* verifying the inequality $d(H_k^*,T) < D$.

Lemma 2 supposes that action A_4 has just been performed, which means, according to the preceding discussion, the robot has got to a point H_k^* such that $d(H_k^*,T) < D$. By taking action A_4, no change is carried out apart from setting the distance D to $d(H_k^*,T)$. Therefore, the robot remains in step 2 continuing thus the contour following process on the obstacle O_i. Algorithm 4.2 provides three different ways for leaving the boundary-following behavior. However, the one associated with step 2.a) can be easily discarded due to the lemma's assumption requiring that $T \notin \partial O_i$. Consequently, only steps 2.b) and 2.c) should be considered. Next, we prove that, if there is a solution to the path-planning problem, before completely covering the contour of O_i, the robot will reach a point Q satisfying conditions C_3 and C_4, or in other words, that the transition to the motion-to-goal behavior will be triggered by step 2.c).

As was remarked in definition 1, all H^* and L^* points appear in pairs with the exception of the potential leave point L_0^*. In fact, there is another special case for the previous general assertion which occurs when the target point is not reachable from S, i.e. when T is located in the interior of an obstacle. Under these circumstances, as shown in figure 4.5, the potential hit point with the maximum subscript is not paired either. It is important to note that, in lemma 2, this second exceptional unpaired situation cannot take place because of assuming that the path-planning problem has, at least, a solution. Therefore, if H_k^* is the potential hit point causing the execution of action A_4, such a point is ensured to go with the potential leave point L_k^*. Furthermore, remember that both points, H_k^* and L_k^*, are guaranteed to be located on the boundary of the obstacle O_i.

Let H_j denote the point where the current contour following process was initiated. Then, we are going to demonstrate that L_k^* is not found on the portion of ∂O_i covered by the robot from H_j to H_k^*. To this end, notice that, before getting to the potential hit point H_k^*, the distance D contains a value greater than $d(H_k^*, T)$. Additionally, $d(L_k^*, T) < d(H_k^*, T)$ so that the potential leave point L_k^* clearly fulfills condition C_3 $\Big(d(L_k^*, T) < D\Big)$. On the other hand, condition C_4 is satisfied by any potential leave point. In consequence, L_k^* meets the two necessary conditions for abandoning the boundary-following behavior. Nevertheless, this behavior is supposed to be active from H_j to H_k^*, which definitely confirms that the robot cannot have gone through the point L_k^* while conducting the aforementioned path.

Bearing in mind the previous fact as well as $L_k^* \in \partial O_i$, it seems obvious that, by continuing following the contour of O_i from H_k^*, the robot will arrive at the potential leave point L_k^* before returning to H_j[2]. At that moment, D will be equal to $d(H_k^*, T)$ and, hence, conditions C_3 and C_4 will be still fulfilled at L_k^* proving thus lemma 2. As a result, algorithm 4.2, by means of step 2.c), will force a transition to the motion-to-goal behavior, which will turn L_k^* into a real leave point.

Finally, let us allow for the up-to-now omitted situation where the robot finds a potential hit point H_l^* which complies with the inequality $d(H_l^*, T) < d(H_k^*, T)$ as moving from H_k^* to L_k^* on ∂O_i. In this context, action A_4 will be executed again and the distance D will accordingly change its current value $\Big(d(H_k^*, T)\Big)$ for $d(H_l^*, T)$. On the other hand, knowing that $d(L_l^*, T) < d(H_l^*, T) < d(L_k^*, T) < d(H_k^*, T)$ based on both our hypothesis and definition 1, it is evident that L_k^* is never going to satisfy condition C_3 after the indicated event. Moreover, considering the pairs $(H_k^*,\ L_k^*)$ and $(H_l^*,\ L_l^*)$, the only point where conditions C_3 and C_4 could be met is L_l^*. Consequently, from the viewpoint of the lemma's proof, neither H_k^* nor L_k^* are really significant once the potential hit point H_l^* has been reached by the robot. Therefore, the situation at hand could be equivalently described by ignoring the achievement of H_k^*, or in other words, by assuming just one execution of action A_4 —at H_l^*. Precisely, this is what was done in the preceding proof of lemma 2 so that such a demonstration comprises this particular case as well. □

Theorem 1. *Algorithm 4.2 converges to the target point.*

Assumption/s. T is reachable from S.

Proof. First of all, imagine that action A_4 is not taken during the whole execution of algorithm 4.2. In these circumstances, lemma 1 establishes that the strategy Bug2+ will behave just like Bug2 —algorithm 4.1—, which is known to guarantee convergence whenever possible.

In short, the proof of termination for algorithm 4.1 —and, in consequence, for algorithm 4.2 by considering the above-mentioned situation— is based on observing that the approach monotonically decreases the distance to the target as revealed by expression 4.3. Notice that in algorithm 4.1, every time a hit point (H_j) is defined, the robot walks around the detected obstacle until getting to a potential leave point closer to T than H_j. Because of that, $d(H_j, T) > d(L_j, T)$. Additionally, after leaving L_j, the robot moves straight towards the target until a new obstacle is found. Since it is assumed that obstacles do not touch

[2] The robot will reach the point L_k^* if no other point is found first satisfying conditions C_3 and C_4. However, in such a case, lemma 2 trivially holds

each other, then $d(L_j, T) > d(H_{j+1}, T)$[3].

$$\begin{aligned} d(L_0,T) \geq d(H_1,T) > d(L_1,T) > d(H_2,T) > \\ d(L_2,T) > d(H_3,T) > d(L_3,T) > \quad \cdots \end{aligned} \tag{4.3}$$

Now, we change the context for the theorem's proof by assuming an execution of action A_4 in algorithm 4.2. According to this, let us go to the instant, previous to performing action A_4, in which the last hit point was created and let H_j be such a point. Until that moment, keeping lemma 1 in mind, it seems clear that the behavior of algorithm 4.2 corresponds with the one of algorithm 4.1 since action A_4 has not been taken yet. Consequently, the distance to the target will have been progressively reduced as indicated by expression 4.4. On the other hand, the robot, after defining H_j and later following the boundary of the sensed obstacle, is supposed to reach a point Q which triggers action A_4. To this end, it is important to note that the point Q should fulfill condition C_3, which means that $D = d(H_j, T) > d(Q, T)$. As a result of doing action A_4, the distance D is updated $\Big(D = d(Q,T)\Big)$ and the robot persists in applying the contour following process. In this situation, lemma 2 ensures finding another point Q' where conditions C_3 and C_4 are met[4]. The point Q', therefore, satisfies that $d(H_j, T) > D = d(Q, T) > d(Q', T)$. When the robot arrives at Q', this point will become the next leave point L_j and the inequality $d(H_j, T) > d(L_j, T)$ will extend expression 4.4 as an evidence of the progress made towards T. Notice that the latter fact confirms that the execution of action A_4 constitutes a step forward regarding the convergence of algorithm 4.2. Finally, from point L_j onwards, algorithm 4.2 will behave again as would be done by algorithm 4.1 because action A_4 is not longer carried out. Without loss of generality, one can imagine such a situation as if algorithm 4.1 were started at L_j with no change of the target point T — additionally, observe that T is reachable from L_j[5]. In view of that, algorithm 4.2 is certainly guaranteed to converge, proving thus theorem 1.

$$\begin{aligned} d(L_0,T) \geq \ d(H_1,T) \ > d(L_1,T) > d(H_2,T) > \\ \cdots \quad > d(L_{j-1},T) > d(H_j,T) \end{aligned} \tag{4.4}$$

To conclude, let us consider the case in which action A_4 is taken more than once while performing algorithm 4.2. This proof is, nevertheless, a trivial generalization of the one previously given which supposed just one execution of action A_4. In order to provide the basic background for such a generalization, let us go back to the moment when L_j was defined in the preceding simple-case proof. Remember that this is the point where the robot abandoned the contour following process after having carried out action A_4 for the first —and last— time. It is important to note that, in algorithm 4.2, leave points do have a special meaning since they can be understood, in some sense, as new starting points. This idea is supported by

[3] This strict inequality does have an exception for the case $j = 0$ owing to the fact that the distance $d(L_0, T)$ may coincide with $d(H_1, T)$. There is no difference between $d(L_0, T)$ and $d(H_1, T)$ when the starting point $(S = L_0)$ is placed on the contour curve of an obstacle

[4] Theorem 1 does not assume, as it is done in lemma 2, that the target (T) cannot be placed on the boundary of the obstacle being currently followed by the robot. For this reason, the robot, by walking around such an obstacle, may find T before the point specified by lemma 2. However, if T is achieved, the path-planning task is, by definition, successfully solved so that theorem 1 holds

[5] If, as assumed herein, T is reachable from S, then T is also reachable from any point of a free-obstacle path starting at S and ending with L_j

the fact that, after defining a leave point (for instance, L_j), algorithm 4.2 will act in exactly the same manner as restarting it and setting $S = L_j$. Consequently, if, from point L_j onwards, action A_4 was taken one more time, this second execution of the action could be interpreted as the first one of a suitably restarted algorithm 4.2. Under these circumstances, it is clear that the proof of convergence for the so-called simple case is valid. Furthermore, knowing that, after performing action A_4, a leave point is always defined according to lemma 2, our reasoning can be extended, in a natural way, to any number of executions of the aforementioned action, which completes the proof of theorem 1. □

Definition 2. Let us introduce the concept of *state* in algorithms 4.1 and 4.2 assuming a situation where the boundary-following behavior is active. Additionally, consider H_j as the point in which the ongoing contour following process was started.

The state of algorithm 4.1 is related to H_j, i.e. to the last-defined hit point. To be more precise, the distance $d(H_j, T)$ determines the state of the algorithm that will be denoted, from now on, as E_{Bug2}. Similarly, the state of algorithm 4.2 —E_{Bug2+}— depends on the so-called distance D whose value is initially set to $d(H_j, T)$ when activating the boundary-following behavior. This means that the states of algorithms 4.1 and 4.2 match each other at point H_j. However, notice that, in algorithm 4.2, E_{Bug2+}, as opposed to E_{Bug2} in algorithm 4.1, may change during the execution of the contour following process. Specifically, E_{Bug2+} is updated after finding a boundary point on the current obstacle satisfying just condition C_3 (and not condition C_4).

Lemma 3. *If algorithms 4.1 and 4.2, while following the contour of the same obstacle, move the robot to a point Q where the equality $E_{Bug2} = E_{Bug2+}$ holds and, later, the state E_{Bug2+} is never altered, then the above-mentioned algorithms will leave the contour following process at the same point L_j. Besides, the resultant paths from Q to L_j will be identical for both algorithms.*

Assumption/s. Algorithms 4.1 and 4.2 share an environment where T is reachable from S. On the other hand, about their configuration, the parameter $pCFD$ is set to the same value.

Proof. In algorithms 4.1 and 4.2, after getting to the point Q, the boundary-following behavior will remain active until finding another point Q' that meets conditions C_1 and C_2 for the former strategy, and conditions C_3 and C_4 for the latter one. The existence of the point Q' is ensured because of requiring the reachability of the target as well as by realizing that, in these circumstances, the convergence to T is always guaranteed as stated by [53] and theorem 1. Therefore, once Q' is found, a transition to the motion-to-goal behavior will take place, and the next leave point L_j will be defined.

Let Q'_{Bug2} and Q'_{Bug2+} represent the points where the transition from the boundary-following behavior to the motion-to-goal behavior is going to occur in, respectively, algorithm 4.1 and algorithm 4.2. From point Q to Q'_{Bug2} / Q'_{Bug2+}, both strategies will follow the contour of the obstacle in the same direction due to the common setting of the parameter $pCFD$. This fact means that if Q'_{Bug2} were equal to Q'_{Bug2+}, there would be no differences in the paths generated by algorithms 4.1 and 4.2 from Q until the end of the contour following process[6]. In addition, the equality $L_j = Q'_{Bug2} = Q'_{Bug2+}$ would hold, proving thus lemma 3.

[6] In the given context, these paths would be different if the robot, before leaving the boundary-following behavior, could go through the point Q'_{Bug2} / Q'_{Bug2+} more than once, and the specific number of times getting to the point Q'_{Bug2} were distinct to the one of Q'_{Bug2+}. Nevertheless, the achievement of a boundary point twice

For that reason, the proof of the lemma is focussed on demonstrating that the points Q'_{Bug2} and Q'_{Bug2+} should coincide each other under the assumption that $E_{Bug2} = E_{Bug2+}$. First of all, remember that Q'_{Bug2} and Q'_{Bug2+} are selected as leave points on the basis of conditions C_1 and C_2 in algorithm 4.1, and conditions C_3 and C_4 in algorithm 4.2. As was discussed in lemma 1, conditions C_2 and C_4 are equivalent. On the other hand, regarding conditions C_1 and C_3, they look for a m-line's point whose distance to the target is less than E_{Bug2} in case of C_1, and less than E_{Bug2+} according to C_3. Consequently, knowing that $E_{Bug2} = E_{Bug2+}$, C_1 and C_3 are the same condition as well. In short, algorithms 4.1 and 4.2 will use the same criterion for deciding to abandon the boundary-following behavior so that Q'_{Bug2} and Q'_{Bug2+} cannot be different points. □

Lemma 4. *If algorithms 4.1 and 4.2, while following the contour of the same obstacle from the same hit point H_j, find, in the given order, the sequence of m-line's points H_k^* and L_l^* such that $d(H_k^*, T) < d(L_l^*, T) < d(H_j, T)$, then the above-mentioned strategies will abandon the boundary-following behavior at different locations. More precisely, the leaving will occur at L_l^* in algorithm 4.1, and at a later point on the obstacle boundary in algorithm 4.2. Therefore, from H_j to L_l^*, the resultant paths will be identical for both planners. This is the only situation in which algorithms 4.1 and 4.2, when sharing H_j, will define a different leave point L_j.*

Assumption/s. Algorithms 4.1 and 4.2 are performed in the same environment with T being reachable from S. Besides, the parameter $pCFD$ is identically set in both strategies.

Proof. In algorithms 4.1 and 4.2, the contour following process is supposed to be started at the same hit point H_j on the same obstacle. Observe that the states E_{Bug2} and E_{Bug2+} are initialized to $d(H_j, T)$ at the moment of defining H_j. Consequently, if such an equality —$E_{Bug2} = E_{Bug2+} = d(H_j, T)$— were satisfied during the whole activation of the boundary-following behaviors, algorithm 4.1 and algorithm 4.2 would leave the contour of the obstacle at an identical point L_j in accordance with lemma 3. However, this coincidence is not pursued by lemma 4, which tries to identify the set of cases where the opposite occurs, i.e. where a different leave point is defined by both strategies. Therefore, the situations of interest for lemma 4 are found in a context requiring that $E_{Bug2} \neq E_{Bug2+}$. This means that, after defining H_j, either the state E_{Bug2} or the state E_{Bug2+} should change before finishing the contour following process of any of the corresponding algorithms. Remember that E_{Bug2} is never altered by algorithm 4.1 being thus equal to $d(H_j, T)$ all the time. On the contrary, in algorithm 4.2, E_{Bug2+} is modified when getting to a point fulfilling condition C_3 but not condition C_4, or in other words, when reaching a potential hit point H_k^* such that $d(H_k^*, T) < E_{Bug2+} = d(H_j, T)$.

In short, the preceding discussion confirms the necessity of achieving a point H_k^* with $d(H_k^*, T) < d(H_j, T)$ in order to make algorithm 4.1 and algorithm 4.2 behave different with regard to leaving the obstacle boundary. It is important to note that the obligation for algorithm 4.2 to pass through H_k^* is also shared by algorithm 4.1. This is due to the fact that both planners choose the same direction —$pCFD$— for following the contour of the obstacle so that their resultant paths will inevitably come together from H_j to H_k^*. At point H_k^*, E_{Bug2+} will be updated by assigning it the value $d(H_k^*, T)$, and E_{Bug2} will keep constant $\Big(= d(H_j, T)\Big)$. Afterwards, algorithms 4.1 and 4.2 will jointly continue the contour following process in search of a proper place where abandoning it. As was highlighted in definition 1,

—or even more times— would necessarily involve a loop around the contour of the obstacle and, accordingly, the execution of the corresponding algorithm would be stopped because of (mis)interpreting that the target is unreachable. Notice that this contradicts the convergence property of algorithms 4.1 and 4.2

the leaving of the boundary-following behavior should always arise at a potential leave point (L^*). Accordingly, let L^*_{Bug2} and L^*_{Bug2+} be the points where such a leaving is going to occur in, respectively, algorithm 4.1 and algorithm 4.2. Notice that the existence of these points is guaranteed because of assuming that the target is reachable. Next, we are intended to analyze the whole set of situations in which L^*_{Bug2} and L^*_{Bug2+} may be different.

As already said, L^*_{Bug2} should meet conditions C_1 and C_2 to become a real leave point in algorithm 4.1. The former condition involves the fulfillment of the expression $d(L^*_{Bug2}, T) < E_{Bug2} = d(H_j, T)$, while the latter one is inherently hold by any potential leave point and, hence, by L^*_{Bug2}, not being thus relevant at all. Something similar happens with L^*_{Bug2+} in algorithm 4.2 where conditions C_3 and C_4 need to be satisfied, which implies that $d(L^*_{Bug2+}, T) < E_{Bug2+} = d(H^*_k, T)$. In the context previously set out, L^*_{Bug2} and L^*_{Bug2+} are supposed to be real leave points (L_j's) since they represent those locations where the boundary-following behavior is going to finish in algorithms 4.1 and 4.2, respectively. Therefore, these points comply with $d(L^*_{Bug2}, T) < d(H_j, T)$ and $d(L^*_{Bug2+}, T) < d(H^*_k, T)$. On the other hand, by resuming the contour following process from H^*_k, algorithm 4.1 and algorithm 4.2 will simultaneously go through the same boundary points. This will remain until one of the strategies stops from circumnavigating the obstacle, i.e. until getting to either L^*_{Bug2} or L^*_{Bug2+}. Imagine that L^*_{Bug2+} is the first point —between the two candidates— to be reached by both planners. As a direct consequence of the preceding fact, the motion-to-goal behavior will take control of the robot in algorithm 4.2. As stated above, L^*_{Bug2+} is a point which fulfills the inequality $d(L^*_{Bug2+}, T) < d(H^*_k, T)$. In addition, remember that $d(H^*_k, T) < d(H_j, T)$. Then, it seems clear that $d(L^*_{Bug2+}, T) < E_{Bug2} = d(H_j, T)$, meaning thus that condition C_1 is met at L^*_{Bug2+}. Furthermore, condition C_2 is also satisfied because of being a potential leave point. Consequently, algorithm 4.1 will abandon the boundary-following behavior at point L^*_{Bug2+} just like algorithm 4.2, or in brief, $L^*_{Bug2} = L^*_{Bug2+}$.

Let us consider now the case where L^*_{Bug2} is reached first —that is, before L^*_{Bug2+}— by algorithms 4.1 and 4.2. As expected, under these circumstances, the leaving will occur for the former strategy. With respect to algorithm 4.2, nevertheless, several things may happen depending on the distance-to-goal relation among the points L^*_{Bug2}, H^*_k, and H_j. We know that $d(L^*_{Bug2}, T) < d(H_j, T)$ and $d(H^*_k, T) < d(H_j, T)$, but no order is given between L^*_{Bug2} and H^*_k. In this sense, only $d(L^*_{Bug2}, T) < d(H^*_k, T)$ and $d(H^*_k, T) < d(L^*_{Bug2}, T)$ are feasible expressions based on definition 1, which states that no m-line's point can be, at once, a potential hit point (H^*) and a potential leave point (L^*). In this way, the equality $d(L^*_{Bug2}, T) = d(H^*_k, T)$ is plainly discarded.

Next, let us examine the behavior of algorithm 4.2 in face of the two possible situations discussed earlier. First of all, assume that $d(L^*_{Bug2}, T) < d(H^*_k, T) = E_{Bug2+}$. In such a case, condition C_3 will be hold at L^*_{Bug2}, just like condition C_4 which is successfully met in all L^*-type points. As a result, the contour following process currently performed by algorithm 4.2 will bring to an end and, accordingly, $L^*_{Bug2+} = L^*_{Bug2}$. Secondly, by supposing that $E_{Bug2+} = d(H^*_k, T) < d(L^*_{Bug2}, T)$, it is obvious that condition C_3 will not be satisfied at L^*_{Bug2}. Thus, irrespective of the fulfillment of condition C_4, no action will be taken by algorithm 4.2 which will continue following the boundary of the obstacle in search of the potential leave point L^*_{Bug2+}. In view of that, notice that $L^*_{Bug2+} \neq L^*_{Bug2}$.

Finally, let us summarize the line of reasoning just developed, which leads to the proof of lemma 4. Two events should arise for making algorithm 4.1 and algorithm 4.2 define a different leave point ($L^*_{Bug2} \neq L^*_{Bug2+}$) when these strategies are carrying out a contour following process initiated from the same location (H_j). On the one hand, the achievement of a potential hit point H^*_k with $d(H^*_k, T) < d(H_j, T)$ in order to impose different internal states

—$E_{Bug2} = d(H_j, T) \neq E_{Bug2+} = d(H_k^*, T)$— on algorithms 4.1 and 4.2. On the other hand, such planners should later get to a potential leave point L_l^* satisfying either condition C_1 or condition C_3 but never both at the same time, allowing thus just the leaving of one of the strategies while the other one continues circumnavigating the boundary of the obstacle. It is important to note that no point exists where condition C_3 is fulfilled alone. In consequence, the aforementioned situation can only be found when L_l^* meets condition C_1 $\Big(d(L_l^*, T) < E_{Bug2} = d(H_j, T)\Big)$ and not condition C_3 $\Big(d(L_l^*, T) > E_{Bug2+} = d(H_k^*, T)\Big)$, or in other words, when L_l^* turns into a real leave point in exclusively algorithm 4.1 ($L_l^* = L_{Bug2}^* = L_j$). This fact means that $d(H_k^*, T) < d(L_l^*, T) < d(H_j, T)$, which concludes the lemma's proof. □

Definition 3. During the activation of the boundary-following behavior, algorithms 4.1 and 4.2 traverse a portion of the contour of a certain obstacle. Next, we formalize the result of such a circumnavigation by introducing the concept of *boundary-following path* in a context where the target point (T) is assumed to be reachable. Observe that, in these circumstances, there is not chance for a complete loop around the obstacle boundary so that the aforesaid boundary-following paths clearly describe open curves. Moreover, they are also simple —i.e. without self-crossings— because of working under the *Jordan curve theorem* [72], which restricts the shape of obstacles —and, consequently, the boundary-following paths as well— in the way pointed out. Let $I = [0, 1]$ mean the unit interval and ∂O_i the contour curve of the obstacle being currently circumnavigated by the robot. Then, the boundary-following path associated with this activation of the contour following process is given by a continuous injective function $\alpha : I \to \partial O_i$. $\alpha(0)$ represents the last-defined hit point, while $\alpha(1)$ indicates the specific location where the boundary-following behavior was finally abandoned.

By way of example, figures 4.6(a) and (b) show the trajectories generated by, respectively, algorithm 4.1 and algorithm 4.2 in a scenario with an intricate obstacle. As can be seen, for the former strategy, the boundary-following behavior was activated on two occasions before converging to T. Accordingly, their corresponding boundary-following paths are denoted as α' and β' $\Big($notice that $\alpha'(0) = H_1'$, $\alpha'(1) = L_1'$, $\beta'(0) = H_2'$, and $\beta'(1) = L_2'\Big)$. Regarding algorithm 4.2, α characterizes the path that was traversed during its first and only execution of the contour following process $\Big(\alpha(0) = H_1$ and $\alpha(1) = L_1\Big)$.

Hereafter, we will usually adopt an alternative way of illustrating a boundary-following path, which differentiates from the one of figures 4.6(a) and (b) in essentially omitting the circumnavigated obstacle. Figure 4.6(c) shows such a new representation for the boundary-following path labeled as α in figure 4.6(b). By comparing figures 4.6(b) and (c), it seems clear that there are more differences between both representations than just the presence / absence of the obstacle. It is important to note that, in figure 4.6(c), all the intersections of the m-line with α are made explicit. Remember that these intersections correspond to a subset of the so-called potential hit points (H^*) and potential leave points (L^*) (refer to definition 1 for further details). In each H_k^* / L_k^* point, additional information is provided: on the one hand, an arrow is traced over the m-line pointing towards the interior of the obstacle; on the other hand, the transformation, if any, of H_k^* / L_k^* into a real hit / leave point (H_j / L_j) is also specified (observe that this transformation takes always place at the endpoints of a boundary-following path); and, lastly, an update of the algorithm's state at H_k^* is designated by writing, in brackets, either E_{Bug2} or E_{Bug2+} depending on the strategy from which the path has been obtained. As was described in definition 2, changes in E_{Bug2} / E_{Bug2+} can only occur in potential hit points. Moreover, when a change arises at H_k^*, the distance $d(H_k^*, T)$ becomes the new state of the algorithm, i.e. $E_{Bug2} = d(H_k^*, T) / E_{Bug2+} = d(H_k^*, T)$.

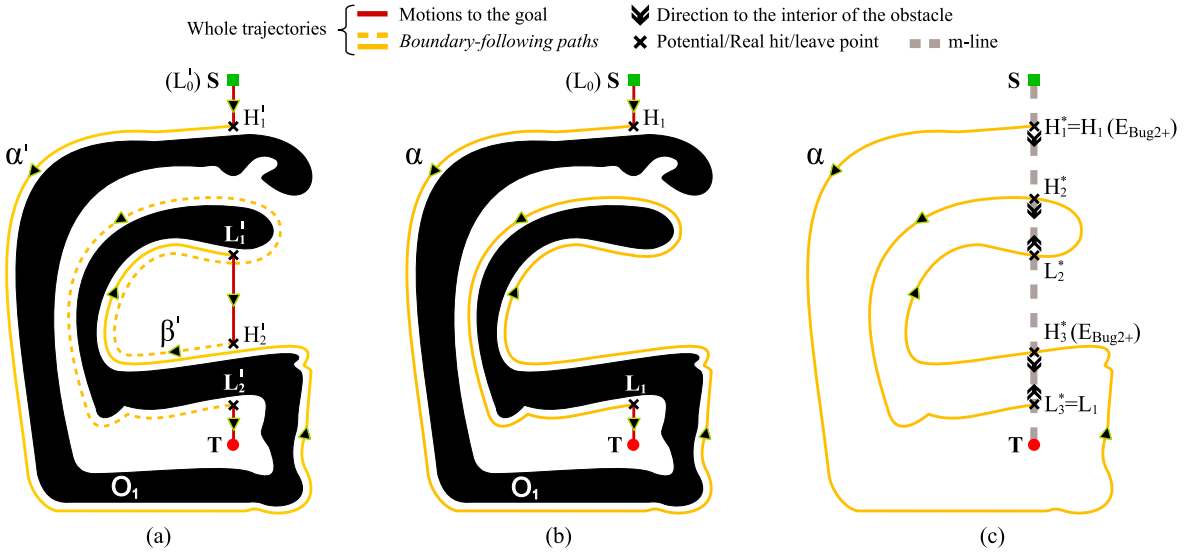


Figure 4.6: Exemplification of the concept of boundary-following path.

Definition 4. Assuming that α is a boundary-following path, let $\Delta^\alpha = \{\delta_1^\alpha, \ldots, \delta_n^\alpha\}$ be a set containing those points of α such that: (1) $\delta_j^\alpha \in H^* \cup L^* \quad \forall j\ 1 \leq j \leq n$ —$n \geq 2$[7]; (2) $\alpha^{-1}(\delta_k^\alpha) < \alpha^{-1}(\delta_{k+1}^\alpha)\ \forall k\ 1 \leq k < n$[8]. In addition, $\alpha_{l,m}$ denotes the segment of α joining δ_l^α and δ_m^α with $1 \leq l < m \leq n$. The curve linked to $\alpha_{l,m}$ is formally described by means of a continuous injective function $\alpha_{l,m} : I \to \partial O_i$, just as was done for α in definition 3. Accordingly, $\alpha_{l,m}(0) = \delta_l^\alpha$, $\alpha_{l,m}(1) = \delta_m^\alpha$, and the rest of the points of the curve are given by $\alpha_{l,m}(j)$ with $j \in (0,1)$. Going ahead with the definition of new terms, $\Delta_{l,m}^\alpha = \{\delta_l^\alpha, \ldots, \delta_m^\alpha\} \subseteq \Delta^\alpha$ —$1 \leq l < m \leq n$—, $\Delta_{l,m}^{\alpha,H^*} = \Delta_{l,m}^\alpha \cap H^*$, $\Delta_{l,m}^{\alpha,L^*} = \Delta_{l,m}^\alpha \cap L^*$, and, finally, the next two conditions are satisfied by $\Theta_{l,m}^\alpha = \{\theta_l^\alpha, \ldots, \theta_m^\alpha\}$: (1) $\Theta_{l,m}^\alpha = \Delta_{l,m}^\alpha$; (2) $d(\theta_j^\alpha, T) > d(\theta_{j+1}^\alpha, T)\ \forall j\ l \leq j < m$. As can be observed, $\Delta_{l,m}^\alpha$ and $\Theta_{l,m}^\alpha$ are the same sets. However, it is important to note that they impose a different order on the elements through the subscript. In more formal words, $\forall \delta_j^\alpha \in \Delta_{l,m}^\alpha$, $\exists!\ \theta_k^\alpha \in \Theta_{l,m}^\alpha \mid \delta_j^\alpha = \theta_k^\alpha$ —and, similarly, $\forall \theta_j^\alpha \in \Theta_{l,m}^\alpha$, $\exists!\ \delta_k^\alpha \in \Delta_{l,m}^\alpha \mid \theta_j^\alpha = \delta_k^\alpha$. See figure 4.7 for a better comprehension of the notation previously introduced.

Now, let $\alpha_{j,k}$ and $\beta_{l,m}$ symbolize two segments of the boundary-following paths α and β, respectively $\Big($observe that $1 \leq j < k \leq |\Delta^\alpha|$ and $1 \leq l < m \leq |\Delta^\beta|\Big)$. Then, $\alpha_{j,k}$ and $\beta_{l,m}$ are said to be *oml-homotopic* when the following hold:

1. $k - j = m - l$ $\Big($or, equivalently, $|\Delta_{j,k}^\alpha| = |\Delta_{l,m}^\beta|\Big)$;
2. $\forall \delta_{j+q}^\alpha \in \Delta_{j,k}^\alpha$, $\delta_{l+q}^\beta \in \Delta_{l,m}^\beta$ $\Big(0 \leq q \leq k - j\ (= m - l)\Big)$, either $\delta_{j+q}^\alpha \in \Delta_{j,k}^{\alpha,H^*}$ and $\delta_{l+q}^\beta \in \Delta_{l,m}^{\beta,H^*}$ or $\delta_{j+q}^\alpha \in \Delta_{j,k}^{\alpha,L^*}$ and $\delta_{l+q}^\beta \in \Delta_{l,m}^{\beta,L^*}$;
3. $\forall \delta_{j+s}^\alpha \in \Delta_{j,k}^\alpha$, $\theta_{j+q}^\alpha \in \Theta_{j,k}^\alpha$, $\delta_{l+s}^\beta \in \Delta_{l,m}^\beta$, $\theta_{l+r}^\beta \in \Theta_{l,m}^\beta$ $\Big(0 \leq q, r, s \leq k - j\ (= m - l)\Big)$, $q = r$ if $\delta_{j+s}^\alpha = \theta_{j+q}^\alpha$ and $\delta_{l+s}^\beta = \theta_{l+r}^\beta$;

[7] A boundary-following path starts at a potential (real) hit point and ends at a potential (real) leave point
[8] α^{-1} indicates the inverse image / preimage of the function $\alpha : I \to \partial O_i$

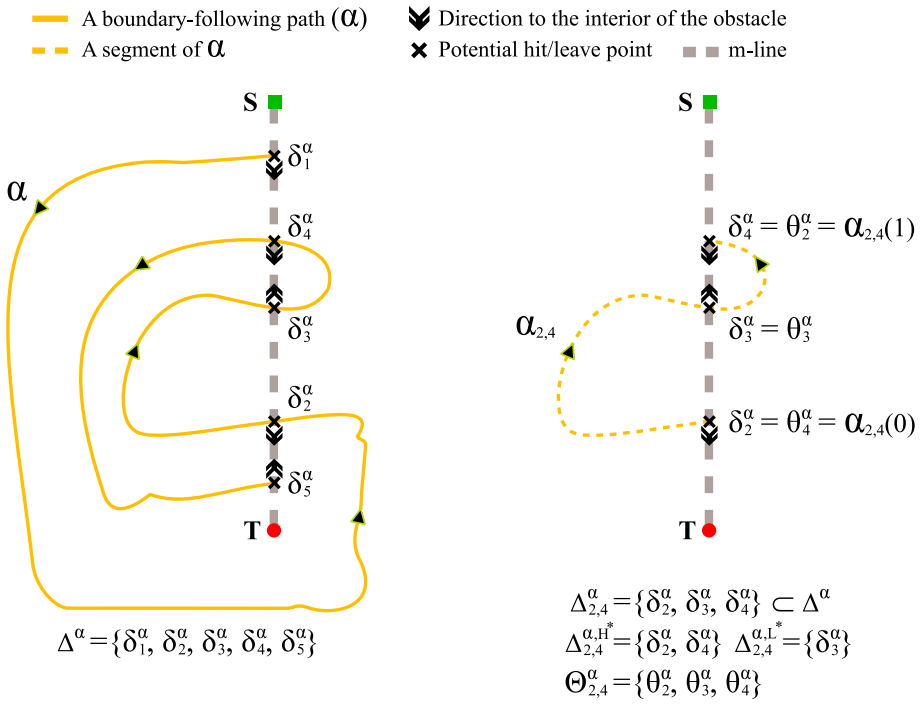


Figure 4.7: Understanding some notation using as an example the boundary-following path of figure 4.6(c). In Δ^α, its elements are enumerated according to the order they are found when following α from the start to the end of the path. On the other hand, in $\Theta_{2,4}^\alpha$, the enumeration is based on the distance to the target (T). Lastly, to properly define the sets $\Delta_{2,4}^{\alpha,H^*}$ and $\Delta_{2,4}^{\alpha,L^*}$, we need to know if a given $\delta_j^\alpha \in \Delta_{2,4}^\alpha$ —$2 \leq j \leq 4$— is either a potential hit point (H^*) or a potential leave point (L^*). In short, $\delta_j^\alpha \in \Delta_{2,4}^{\alpha,H^*}$ if the arrow drawn at δ_j^α is pointing towards T; otherwise, $\delta_j^\alpha \in \Delta_{2,4}^{\alpha,L^*}$.

4. $\forall q\ 0 \leq q < k-j\ (= m-l)$, there exists a continuous function $h : I \times I \rightarrow \mathbb{R}^2$ which can deform the segment $\alpha_{j+q,j+q+1}$ into $\beta_{l+q,l+q+1}$. More precisely, the function h should comply with:

   4.1. $\forall r\ 0 \leq r \leq 1$, $h(0,r) = \alpha_{j+q,j+q+1}(r)$ and $h(1,r) = \beta_{l+q,l+q+1}(r)$;

   4.2. $\forall s\ 0 \leq s \leq 1$, the curve $\gamma_s(r) = h(s,r)$ with $0 < r < 1$ does not cross the m-line (ST) at any point. Besides, $\forall \epsilon > 0\ \exists r_0 \in (0,\epsilon)$ and $r_1 \in (1-\epsilon, 1) \mid \gamma_s(r_0)$ and $\gamma_s(r_1) \notin ST$;

   4.3. $\forall s\ 0 \leq s \leq 1$, $h(s,0)$ and $h(s,1) \in ST \setminus \{S,T\}$.

With the aim of discussing about the *oml-homotopy* requirements listed above, figure 4.8 considers several pairs of segments $\alpha_{j,k}$ and $\beta_{l,m}$. Notice that the segments forming each pair are going to be part of the boundary-following paths α and β illustrated in figures 4.8(a) and (b), respectively. Both boundary-following paths are supposed to be generated by the algorithm 4.2 with its parameter *pCFD* set to *right*. Let $\alpha_{1,9} = \alpha$ and $\beta_{1,11} = \beta$ define the first pair of segments to be checked for the fulfillment of the four conditions associated with the concept of oml-homotopy. In such a pair, condition 1 is not met because $k - j = 9 - 1 \neq m - l = 11 - 1$. Therefore, $\alpha_{1,9}$ and $\beta_{1,11}$ are not *oml-homotopically* equivalent, irrespective of the rest of conditions whose analysis is no longer needed. Now, focussing our attention on the segments $\alpha_{4,6}$ and $\beta_{3,5}$, it seems evident that this new pair satisfies condition 1 ($k - j = 6 - 4 = m - l = 5 - 3$), but not condition 2. In plain words, this second condition involves the classification of the elements in the sets $\Delta^{\alpha}_{4,6}$ and $\Delta^{\beta}_{3,5}$ into either potential hit points $\left(\Delta^{\alpha,H^*}_{4,6} \,/\, \Delta^{\beta,H^*}_{3,5}\right)$ or potential leave points $\left(\Delta^{\alpha,L^*}_{4,6} \,/\, \Delta^{\beta,L^*}_{3,5}\right)$, as shown in figure 4.8(c). More precisely, condition 2 holds when the element with subscript $4+q$ in $\Delta^{\alpha}_{4,6}$ and the one with subscript $3+q$ in $\Delta^{\beta}_{3,5}$ are identically classified for all q —$0 \leq q \leq 2$. Assuming that $q = 0$, we can realize by observing figure 4.8(c) that $\delta^{\alpha}_4 \in \Delta^{\alpha}_{4,6}$ and $\delta^{\beta}_3 \in \Delta^{\beta}_{3,5}$ are not in agreement with each other, in the sense that the former element is a potential hit point while the latter one is a potential leave point. Next, let $\alpha_{4,6}$ and $\beta_{6,8}$ be the third pair of segments taken into consideration. On this occasion, the pair fulfills conditions 1 and 2, but not condition 3 which forces the elements in $\Delta^{\alpha}_{4,6}$ and $\Delta^{\beta}_{6,8}$ to be distributed on the m-line in the same relative way. To this respect, the set $\Theta^{\alpha}_{4,6} \,/\, \Theta^{\beta}_{6,8}$ provides the position that an element in $\Delta^{\alpha}_{4,6} \,/\, \Delta^{\beta}_{6,8}$ occupies when traversing the m-line from S to T. Specifically, if $\delta^{\alpha}_{4+q} \in \Delta^{\alpha}_{4,6} \,/\, \delta^{\beta}_{6+q} \in \Delta^{\beta}_{6,8}$ and $\theta^{\alpha}_{4+r} \in \Theta^{\alpha}_{4,6} \,/\, \theta^{\beta}_{6+r} \in \Theta^{\beta}_{6,8}$ represent the same point in $\mathbb{R}^2$ —$0 \leq q, r \leq 2$—, then the position of $\delta^{\alpha}_{4+q} \,/\, \delta^{\beta}_{6+q}$ on the m-line is $r + 1$. As can be seen in figure 4.8(d), $\delta^{\alpha}_{4+0} \in \Delta^{\alpha}_{4,6} = \theta^{\alpha}_{4+0} \in \Theta^{\alpha}_{4,6}$ and $\delta^{\beta}_{6+2} \in \Delta^{\beta}_{6,8} = \theta^{\beta}_{6+1} \in \Theta^{\beta}_{6,8}$ so that the positions of δ^{α}_{4+0} and δ^{β}_{6+2} (referred to as $S \rightarrow T$-based positions from now on) are, respectively, 1 and 2. Condition 3 demands that the element with subscript $4 + s$ in $\Delta^{\alpha}_{4,6}$ and the one with subscript $6 + s$ in $\Delta^{\beta}_{6,8}$ have the same $S \rightarrow T$-based position for all s —$0 \leq s \leq 2$. Unfortunately, this is not true when either $s = 1$ or $s = 2$. Just to demonstrate one of these failing cases, $\delta^{\alpha}_{4+1} \in \Delta^{\alpha}_{4,6} = \theta^{\alpha}_{4+1} \in \Theta^{\alpha}_{4,6}$ and $\delta^{\beta}_{6+1} \in \Delta^{\beta}_{6,8} = \theta^{\beta}_{6+2} \in \Theta^{\beta}_{6,8}$ for $s = 1$, which means that the $S \rightarrow T$-based positions of δ^{α}_{4+1} and δ^{β}_{6+1} are certainly different ($2 \neq 3$). Finally, $\alpha_{1,3}$ and $\beta_{1,3}$ form the last pair of segments to be examined. In short, all conditions are successfully met by the above-mentioned pair with the only exception of condition 4, which imposes some restrictions on $\alpha_{1,3}$ and $\beta_{1,3}$ based on the idea of *primitive* segment. To be clear, a primitive segment corresponds to a portion of $\alpha_{1,3} \,/\, \beta_{1,3}$ connecting two consecutive points of $\Delta^{\alpha}_{1,3} \,/\, \Delta^{\beta}_{1,3}$ (observe

that the preceding / following relation of the elements in the sets $\Delta^{\alpha}_{1,3}$ and $\Delta^{\beta}_{1,3}$ is given by the subscript). Consequently, the expression $\alpha_{1+q,1+q+1} \,/\, \beta_{1+q,1+q+1}$ with $0 \leq q < 2$ characterizes every primitive segment of $\alpha_{1,3} \,/\, \beta_{1,3}$. Condition 4 essentially requires that $\alpha_{1+q,1+q+1}$ can be continuously deformed into $\beta_{1+q,1+q+1}$ for all q. Besides, during such a deformation, no point should cross the m-line (ST) and the endpoints of all intermediate curves between $\alpha_{1+q,1+q+1}$ and $\beta_{1+q,1+q+1}$ should remain fixed to ST —to be more precise, from one curve to another, the position of the endpoints may change, but on condition that they do belong to ST[9]. Figures 4.8(e) and (f) try to deform, in the way explained before, $\alpha_{1+q,1+q+1}$ into $\beta_{1+q,1+q+1}$ for each possible value of q. Particularly, when $q = 0$, any deformation between the aforesaid primitive segments —$\alpha_{1,2}$ and $\beta_{1,2}$— results in failure since the m-line is inevitably crossed by, at least, one intermediate curve as exemplified in figure 4.8(e). In favor of a better understanding, imagine the deformation process as if $\alpha_{1+q,1+q+1}$ were an elastic band with its ends tied to the m-line in such a manner that they can be slipped up and down along ST. According to this, $\beta_{1+q,1+q+1}$ should be obtained by suitably stretching the $\alpha_{1+q,1+q+1}$-based elastic band. On the other hand, when $q = 1$, $\alpha_{2,3}$ can become $\beta_{2,3}$ through a full-compliance deformation (see figure 4.8(f)). This fact, nevertheless, is not enough for satisfying condition 4 given that this condition requires that there is no case that fails; this requirement is, hence, not fulfilled because of the case of $q = 0$.

To conclude, it is important to highlight that there are some relevant differences between the concept of homotopy which has just been introduced and the one given in [72]. Mainly, they differ in that we allow two curves / segments with different endpoints to be homotopic as well as in considering the m-line as the only obstacle that restricts the deformation of one curve / segment into the other. In order to avoid confusion with the original concept ([72]), the prefix *oml* has been used for referring to our particular kind of homotopy —*oml* stands for open m-line.

Lemma 5. *Let α be the boundary-following path hypothesized in lemma 4 which joins $H_j = \alpha(0)$ to $L^*_l = \alpha(1)$ and, in between, passes through H^*_k with $d(H^*_k, T) < d(L^*_l, T) < d(H_j, T)$. Then:*

1. *$\Delta^{\alpha} = \{\delta^{\alpha}_1 = H_j, \ \ldots \ , \delta^{\alpha}_n\}$ with $n \geq 3$;*
2. *The segment $\alpha_{n-2,n}$ is oml-homotopic to one of the curves found in figure 4.9;*
3. *δ^{α}_{n-2} is a potential hit point ($\in H^*$), $\delta^{\alpha}_{n-1} = H^*_k$, and $\delta^{\alpha}_n = L^*_l$.*

Assumption/s. The same assumptions as in lemma 4.

Proof. Before starting the proof of lemma 5, let us see how two of the most important features of a *primitive* boundary-following path segment —hereafter, briefly referred to as primitive segment— are related to each other. More exactly, the study is focussed on revealing the influence that the shape of a primitive segment has on the $H^* \,/\, L^*$-based labeling of its endpoints. In this sense, notice that an endpoint is labeled as H^* when represents a potential hit point, while the label L^* is used otherwise, i.e. when the endpoint denotes a potential leave point (look at definition 1 for a complete description of the requirements that a point should fulfill to be considered a potential hit / leave point). On the other hand, regarding the concept of primitive segment, let α' be a generic boundary-following path with $\Delta^{\alpha'} = \{\delta^{\alpha'}_1, \ldots, \delta^{\alpha'}_{n'}\}$.

[9] Additionally, notice that the endpoints of the entire set of curves involved into the deformation process —including $\alpha_{1+q,1+q+1}$ and $\beta_{1+q,1+q+1}$— are allowed to be located neither at S nor at T

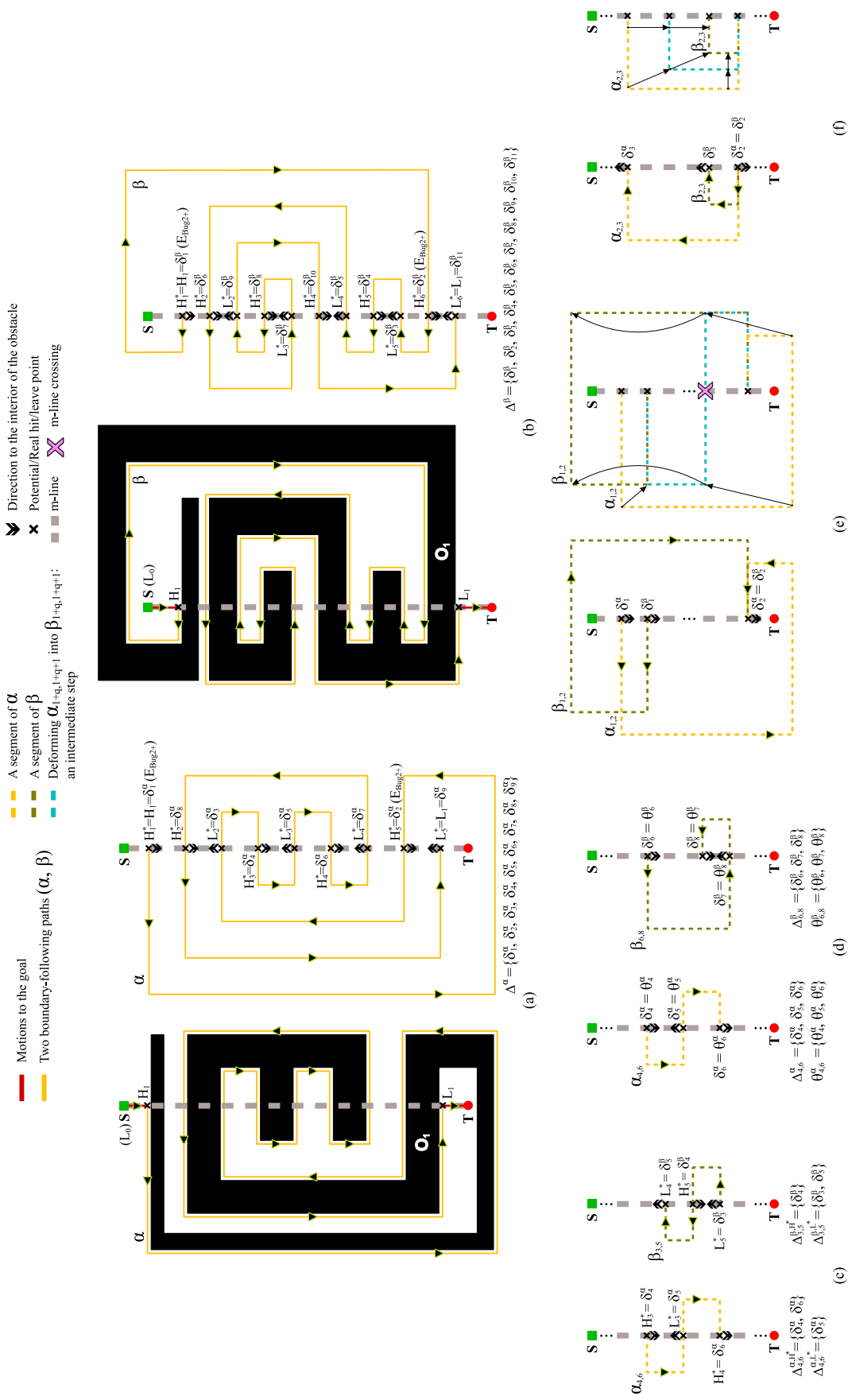


Figure 4.8: Some examples relating to the requirements that two boundary-following path segments should accomplish to be *oml-homotopic*, i.e. to be equivalent from a topological point of view.

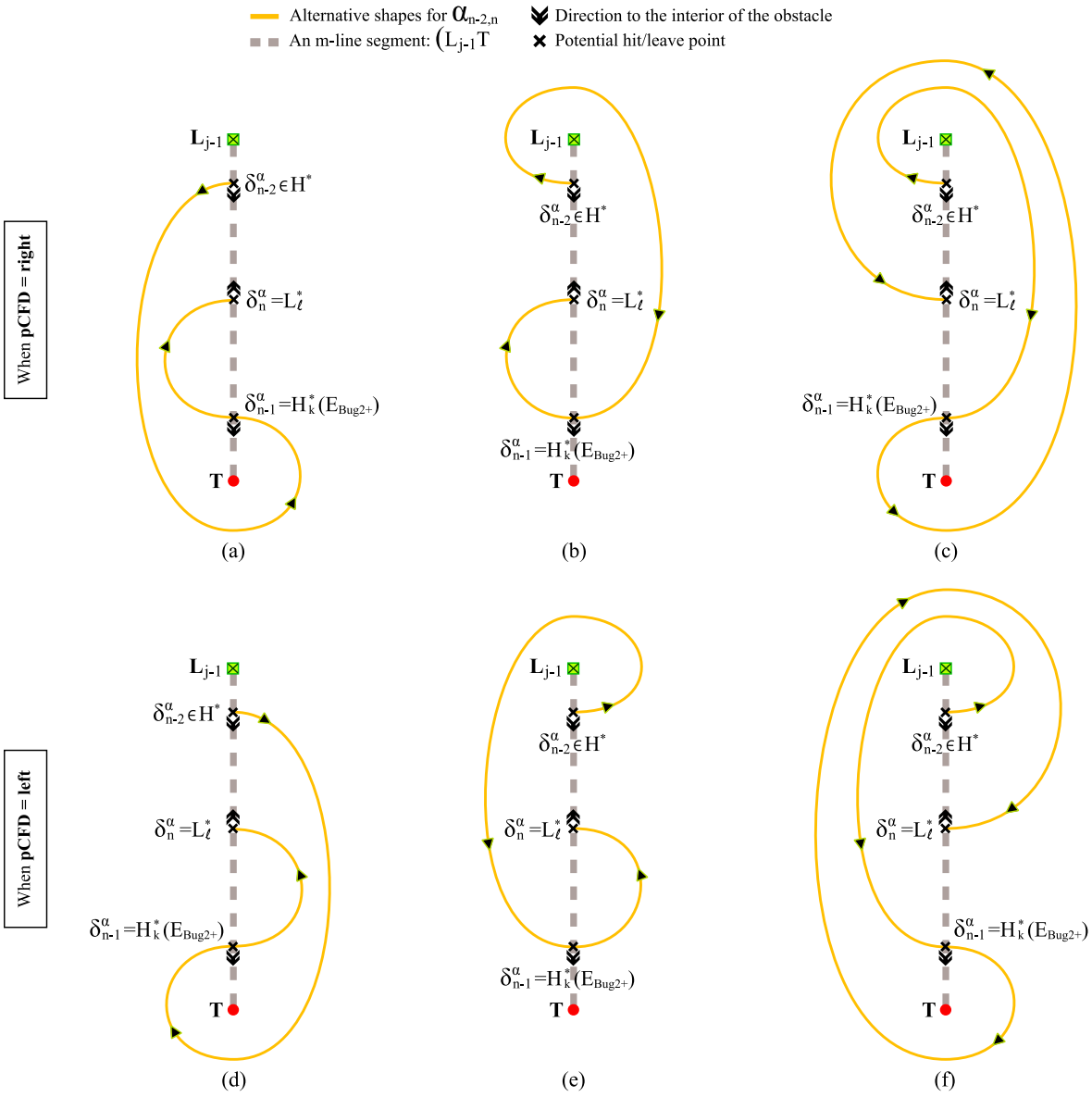


Figure 4.9: Reducing the possible shapes of the segment $\alpha_{n-2,n}$ to just six different cases, which are divided into two groups depending on the direction chosen for traversing the contour of the obstacles —either $pCFD = left$ or $pCFD = right$.

Thus, as pointed out in definition 4, the symbol $\alpha'_{q,q+1}$ —$1 \leq q < n'$— identifies a primitive segment of α'. Informally speaking, the pieces resulting from cutting α' in each point where the m-line is crossed —i.e. in each $\delta_r^{\alpha'} \in \Delta^{\alpha'}$ such that $1 < r < n'$— correspond to all primitive segments of the partitioned boundary-following path.

Next, according to the preceding discussion, let $\alpha'_{q,q+1}$ designate a particular primitive segment of α', specifically the one connecting $\delta_q^{\alpha'} \in \Delta^{\alpha'}$ to $\delta_{q+1}^{\alpha'} \in \Delta^{\alpha'}$. In these circumstances, we state that if $\alpha'_{q,q+1}$ is oml-homotopic to any curve of figure 4.10(a), then the labeling of the endpoints $\delta_q^{\alpha'}$ and $\delta_{q+1}^{\alpha'}$ is ensured not to coincide. Figure 4.10(b) demonstrates the previous fact by performing an exhaustive analysis of cases, which essentially consists in the following: one by one, $\alpha'_{q,q+1}$ adopts the shape of the four curves shown in figure 4.10(a) —this undoubtedly makes $\alpha'_{q,q+1}$ equivalent / oml-homotopic to such curves; at the same time, for each different shape of $\alpha'_{q,q+1}$, the two possible characterizations of $\delta_q^{\alpha'}$ as either a potential hit point (H^*) or a potential leave point (L^*) are taken into account; finally, each of the eight resultant cases is suitably treated, meaning that the label of $\delta_{q+1}^{\alpha'}$ is inferred and also checked to be opposite to the one claimed for $\delta_q^{\alpha'}$. Regarding the above-mentioned inference process, it is important to note that it exploits the representation given to both H^* and L^* points with the already-known purpose of determining the appropriate labeling of $\delta_{q+1}^{\alpha'}$. As said by definition 3, these points are symbolized using a directional arrow, which is always aligned with the m-line (ST). This arrow allows to easily distinguish between H^* and L^* points by observing its pointing direction: in short, a potential hit point is recognized when the arrow is pointing at T, while the opposite —thus, the arrow is pointing at S— suggests the existence of a potential leave point. As one could expect, the earlier criterion for differentiating H^* and L^* points is not arbitrary at all, but it is based on the real essence of the terms *hit* and *leave*. In simple words, an arrow does point at the ST-restricted direction in which the obstacle containing the associated potential hit / leave point lies. To this respect —and going back in the text—, figure 4.6(c) provides several illustrative examples (see also figure 4.6(b) to know the precise shape of the obstacle that is involved in such examples). As can be appreciated, at any point labeled as $H_k^* \,/\, L_k^*$, we can get directly inside the corresponding obstacle by following the direction specified by the arrow (or, from another point of view, we can consistently affirm that the direction of the arrow, after a U-turn, locally moves us away from the obstacle). Once having explained the notions needed for a well understanding of the so-called inference process, let us discuss how it is applied. To this end, concerning the case depicted in the left-upper corner of figure 4.10(b), we are going to find out the only feasible option for the labeling of $\delta_{q+1}^{\alpha'}$. First of all, notice that $\delta_q^{\alpha'}$ is supposed to be a potential hit point since its arrow is pointing at T. Moreover, the curve joining $\delta_q^{\alpha'}$ to $\delta_{q+1}^{\alpha'}$ —$\alpha'_{q,q+1}$— constitutes a portion of the boundary of a certain obstacle. Remember that, in accordance with the *Jordan curve theorem* [72], the whole boundary of an obstacle is guaranteed to define a simple closed curve, which implies that it divides the plane $\left(\mathbb{R}^2\right)$ into just two regions commonly referred to as the *inside* and the *outside*. In view of that, the arrow linked to a potential hit / leave point can be alternatively interpreted as a graphical way of showing the direction towards the inside region of the obstacle to which the point belongs to. Consequently, knowing the labeling of $\delta_q^{\alpha'}$, we are able to identify which side of the curve $\alpha'_{q,q+1}$ actually corresponds to the interior / exterior of the obstacle, as done in figure 4.10(b) for the case at hand. Lastly, keeping this information in mind, the pointing direction of the arrow at $\delta_{q+1}^{\alpha'}$ is trivially obtained. As expected, $\delta_{q+1}^{\alpha'}$ disagrees with $\delta_q^{\alpha'}$ because of being a potential leave point. To conclude, observe that there is no case in figure 4.10(b) where $\delta_q^{\alpha'}$ and $\delta_{q+1}^{\alpha'}$ are both potential

hit points / potential leave points, proving so the formulated statement.

At this moment, by following the same line of reasoning which was previously developed, we can additionally state that:

- If $\alpha'_{q,q+1}$ is oml-homotopic to any curve of figure 4.11(a) —these curves are characterized by going around either S or T—, the labeling of the endpoints $\delta_q^{\alpha'}$ and $\delta_{q+1}^{\alpha'}$ are ensured to be identical.

- Similarly, if $\alpha'_{q,q+1}$ is oml-homotopic to any curve of figure 4.11(b) —these curves differ from all others in that they go first around $S\,/\,T$ and, later, around $T\,/\,S$—, the labeling of the endpoints $\delta_q^{\alpha'}$ and $\delta_{q+1}^{\alpha'}$ are guaranteed not to coincide, just as happened for the set of curves represented in figure 4.10(a).

By way of example, figure 4.11(c) presents some situations which clearly support the above-mentioned conditional assertions. In each of the four settings considered, after having defined $\delta_q^{\alpha'}$ as either an H^* or L^* point, the label of $\delta_{q+1}^{\alpha'}$ is decided by means of the already-known inference process.

As a closing remark for the preliminary concepts involved in the proof of lemma 5, it is important to highlight that figure 4.10(a), and figures 4.11(a) and (b) illustrate all possible shapes of the primitive segment $\alpha'_{q,q+1}$, or in other words, any valid shape for $\alpha'_{q,q+1}$ is oml-homotopically equivalent to one of the curves shown in the aforesaid figures. Such a reduced set of solutions results from discarding those shapes of $\alpha'_{q,q+1}$ that fail in satisfying the *Jordan curve theorem*. In this sense, $\alpha'_{q,q+1}$ is part of the contour of an obstacle, i.e. of a *Jordan* curve, and, therefore, it should be simple —without self-crossings. This requirement, which is inherent to the idea of primitive segment, is merely fulfilled by the group of curves found in figures 4.10(a) and 4.11(a,b). Lastly, just to give an example of a candidate shape for $\alpha'_{q,q+1}$ that was rejected on the basis of the *Jordan curve theorem*, figure 4.12(a) draws a curve which goes, in order, around S, T, and S before defining the endpoint $\delta_{q+1}^{\alpha'}$. As can be seen, there is no other alternative for a curve with the preceding topological description than to cross itself at least once.

Lemma 4 contemplates a situation in which algorithm 4.1 and algorithm 4.2, while circumnavigating the contour of the same obstacle, share a path that joins the points H_j and L_l^*, and in-between passes through H_k^* with $d(H_k^*, T) < d(L_l^*, T) < d(H_j, T)$. In this context, H_j is a real hit point which marks the place where such a contour following process / boundary-following behavior was initiated in both planners. It is important to note that, prior to performing the alluded boundary-following behavior, the other strategic component of the algorithms 4.1 and 4.2 was active. We are referring to the so-called motion-to-goal behavior which, as pointed out in section 4.1.3, is intended to get closer to the target (T) by moving along the m-line (ST) until an obstacle definitely impedes to advance further. Consequently, a segment of ST is traversed as a result of the operation of this behavior. Let XY^{mtg} generically denote one of these segments, being X and Y two different m-line's points with $d(X,T) > d(Y,T)$. In short, XY^{mtg} means that the related activation of the motion-to-goal behavior did achieve progress from X to Y by following a straight-line path. Now, we are going to characterize the XY^{mtg}-type segment associated with the execution of the motion-to-goal behavior that immediately precedes the aforementioned contour following process — the one involved in lemma 4. Under these circumstances, it seems obvious that H_j defines the point where an obstacle was found, i.e. where the progress towards T brought to an end. Hence, $Y = H_j$. Additionally, observing that, in algorithm 4.1 and algorithm 4.2, the starting

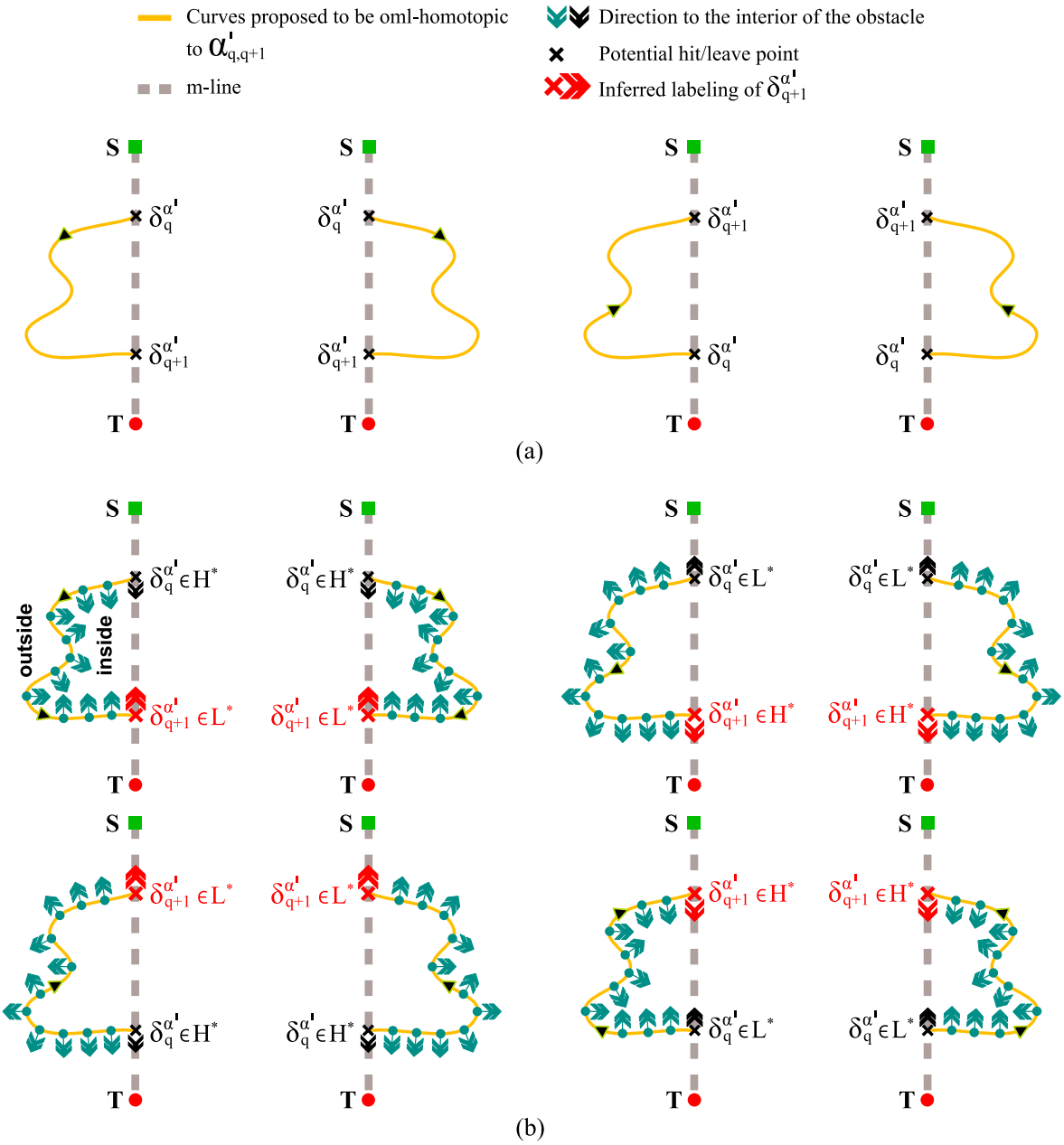


Figure 4.10: Going deeply into details on two key properties of a primitive segment: shape versus labeling of the endpoints.

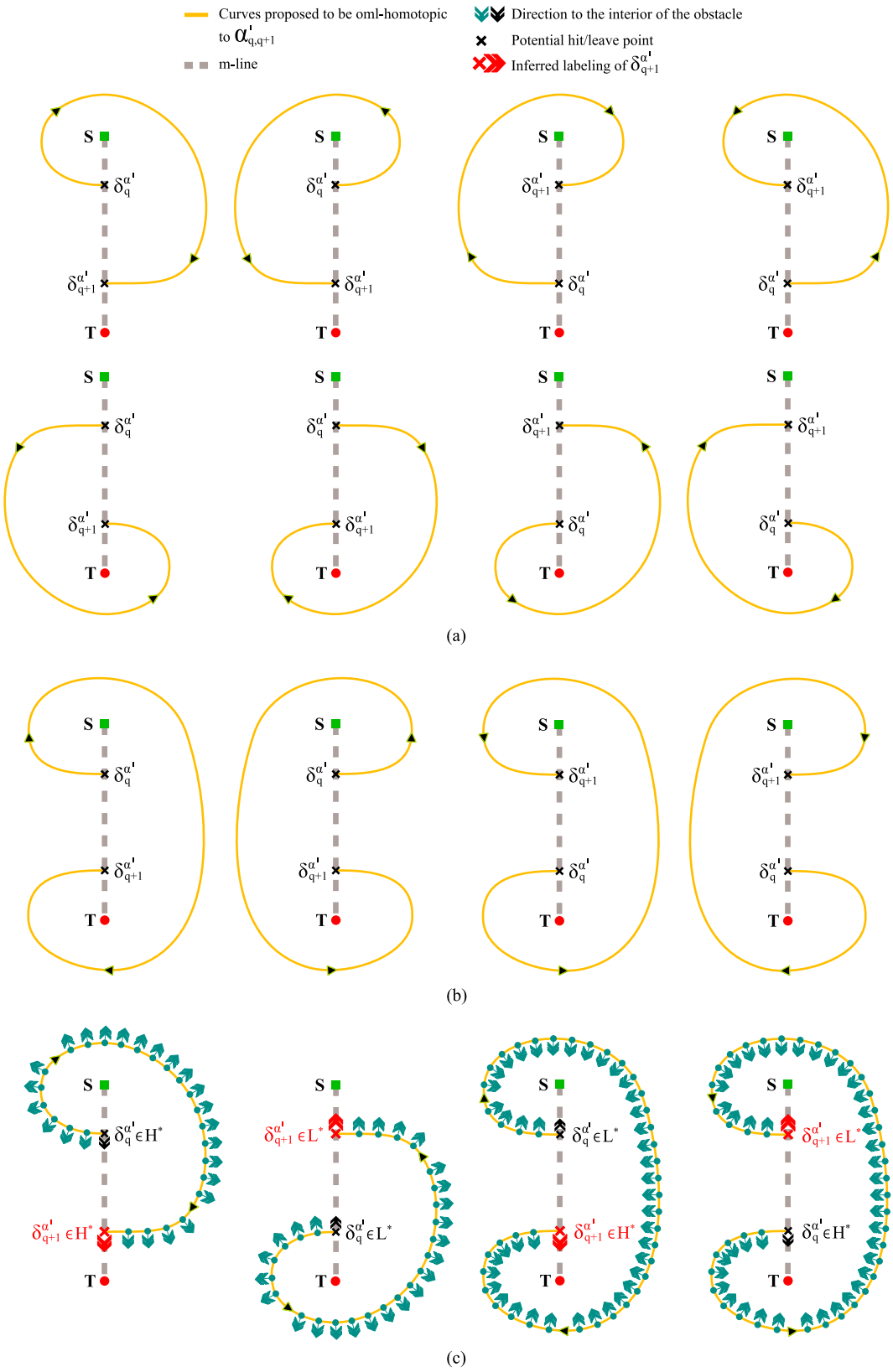


Figure 4.11: More about two key properties of a primitive segment: shape versus labeling of the endpoints.

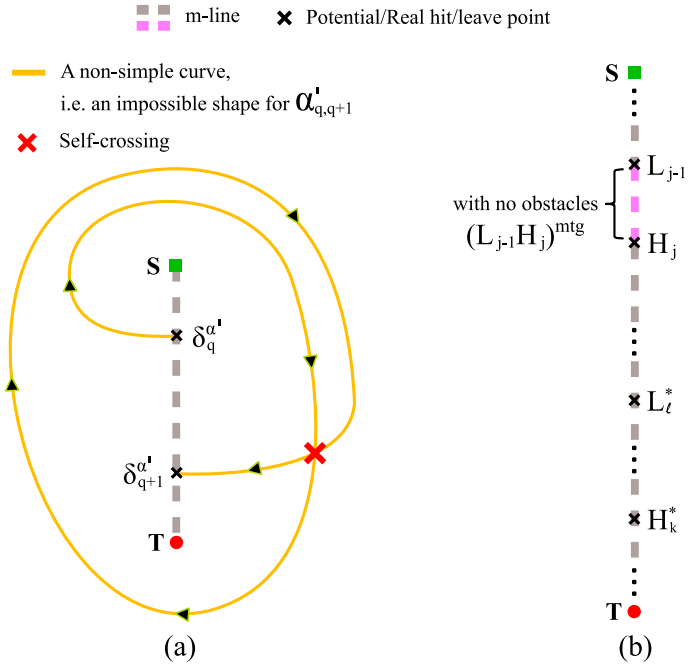


Figure 4.12: (a) an unsuitable shape for the primitive segment $\alpha'_{q,q+1}$ because of not being a simple curve; (b) some inquiries about how the boundary-following path linked to lemma 4 extends before reaching H_j.

of any action requiring some direct motion to the goal always lies on a real leave point as well as the subscript of this L-labeled point can be simply obtained by subtracting one from the subscript of Y $(= H_j)$, then $X = L_{j-1}$. Thus, the segment $L_{j-1}H_j^{mtg}$ constitutes the most recent path preceding the one that connects H_j to L_l^*. Figure 4.12(b) graphically summarizes the major issues to be kept in mind hereafter from the previous discussion. As can be seen, the inequality $d(H_k^*, T) < d(L_l^*, T) < d(H_j, T) < d(L_{j-1}, T)$ holds, making thus evident the relative position of these points over the m-line. On the other hand, the segment $(L_{j-1}H_j)^{mtg}$ —notice that parentheses are used for describing an XY^{mtg}-type segment which is fully *open*, or, in plain words, a segment which does not include any of its endpoints— identifies a path in $\mathbb{R}^2$ that is ensured to be free of obstacles according to the well-known operation of the motion-to-goal behavior.

In the following, we assume that algorithm 4.1 and algorithm 4.2 were both configured with the parameter $pCFD = right$ when moving from H_j to L_l^*, i.e. when producing the boundary-following path referred to as α. The forthcoming analysis trying to infer, under the given assumption, all possible shapes of α —and, in consequence, all possible shapes of any part of it such as $\alpha_{n-2,n}$— can be effortlessly revised for the case being omitted which considers $pCFD = left$. As required by lemma 5 and illustrated in figure 4.9(a), the concerned $pCFD_{right}$-based analysis should be focussed on determining the shape of $\alpha\,/\,\alpha_{n-2,n}$ with respect to a portion of the m-line expressed by $(L_{j-1}T$ —in line with the notation introduced above for XY^{mtg}-type expressions, the use of parentheses in XY-type expressions[10] indicates that the corresponding straight-line segment is open, but just partially on this occasion, because of lacking of a closing parenthesis, which means that the segment does not contain its

[10] In essence, XY and XY^{mtg}-type expressions differ each other in that the former ones symbolize generic straight-line segments and not necessarily paths derived from the execution of the motion-to-goal behavior

endpoint L_{j-1}, but contains, however, T. Observe that, by definition, the boundary-following path α crosses ($L_{j-1}T$ in several points —to be precise, at least in H_j, H_k^*, and L_l^*. This fact clearly guarantees the existence of the primitive boundary-following path segments $\alpha_{1,2}$ and $\alpha_{2,3}$. Accordingly, and as a first step to the proof of lemma 5, let us find out which shapes of those ones that are feasible for a generic primitive segment designed by $\alpha'_{q,q+1}$ continue being valid when dealing with the particular features of $\alpha_{1,2}$. In this sense, figures 4.10 and 4.11 depict every single shape of $\alpha'_{q,q+1}$, or in other words, they provide an initial *oml-homotopy*-based model for the $\alpha_{1,2}$'s shapes. This wide set of solutions, nevertheless, can be significantly reduced until obtaining the one shown in figure 4.13(b) by taking into account the following specific details on $\alpha_{1,2}$: (1) $\alpha'_{q,q+1}(0) = \alpha_{1,2}(0) = \alpha(0) = H_j$, or equivalently, $\delta_q^{\alpha'} = \delta_1^{\alpha} = H_j$; (2) from $H_j \,/\, \alpha_{1,2}(0)$, the contour of the detected obstacle is supposed to be followed in right direction —remember that $pCFD = right$—, which implies that $\alpha_{1,2}$ should go on the right[11] side of the m-line as exemplified in figure 4.13(a), discarding thus half of the potential solutions. Next, the suitability of each of the shapes of $\alpha_{1,2}$ suggested in figure 4.13(b) will be discussed in depth —notice that, as required previously, these shapes are given using as a reference the m-line's segment ($L_{j-1}T$, and not ST:

- *About cases 1 and 7.* As demonstrated earlier when introducing the preliminary notions for the proof of lemma 5, by knowing both the precise shape of a primitive segment and the $H^* \,/\, L^*$-based labeling of one of its endpoints, we are able to determine what kind of potential point —H^* or L^*— the other endpoint is. To this respect, but paying special attention to $\alpha_{1,2}$, the shape of that curve is supposed to be the one of cases 1 and 7. In addition, the endpoint $\delta_1^{\alpha} = \alpha_{1,2}(0)$ belongs to H^* because $\delta_1^{\alpha} = H_j$. Hence, the unknown $H^* \,/\, L^*$-based labeling of $\delta_2^{\alpha} = \alpha_{1,2}(1)$ can be actually found out for the cases at hand, as it is done in figure 4.13(b). Observe that, in accordance with the aforementioned figure, the endpoint δ_2^{α} is a potential leave point —$\in L^*$— in cases 1 and 7. As said by definition 1, an L^*-labeled point such as δ_2^{α} identifies a position in the environment where algorithms 4.1 and 4.2 may decide to abandon an ongoing contour following process. Specifically, the leaving arises when conditions C_1 and C_2 in algorithm 4.1, and conditions C_3 and C_4 in algorithm 4.2 are met. Regarding the former planner, let us see if conditions C_1 and C_2 hold at δ_2^{α} in the current context. On the one hand, condition C_1 demands finding a point such that the distance to the target is less than the one from $\alpha(0)$ ($= \alpha_{1,2}(0) = H_j$, denoting thus the position where the boundary-following path α —or consistently, its first primitive segment $\alpha_{1,2}$— was started). On the basis of figure 4.13(b), the expression $d(\delta_2^{\alpha}, T) < d(\alpha(0), T) = d(H_j, T) = E_{Bug2}$ can be inferred for cases 1 and 7, which definitely means that condition C_1 is satisfied at δ_2^{α}. On the other hand, the same occurs with condition C_2, whose fulfillment comes from the potential-leave-point nature of δ_2^{α}. In short, δ_2^{α} does meet the two necessary conditions for algorithm 4.1 to abandon, in such a point, the contour following process linked to α. However, under lemma 5, algorithm 4.1 is assumed to fully traverse α —as pointed out before, this boundary-following path consists, in its up-to-now realized form, of the primitive segments $\alpha_{1,2}$ and $\alpha_{2,3}$—, and not just a part of it —$\alpha_{1,2}$. This contradiction, without loss of generality, allows us not to be concerned about cases 1 and 7.

- *About cases 2, 4, 6, and 8.* As remarked in figure 4.12(b), the straight-line segment $(L_{j-1}H_j)^{mtg}$ corresponds to a path without obstacles. In contrast, the primitive segment $\alpha_{1,2}$ represents a portion of the boundary of a certain obstacle so that

[11] looking from S to T

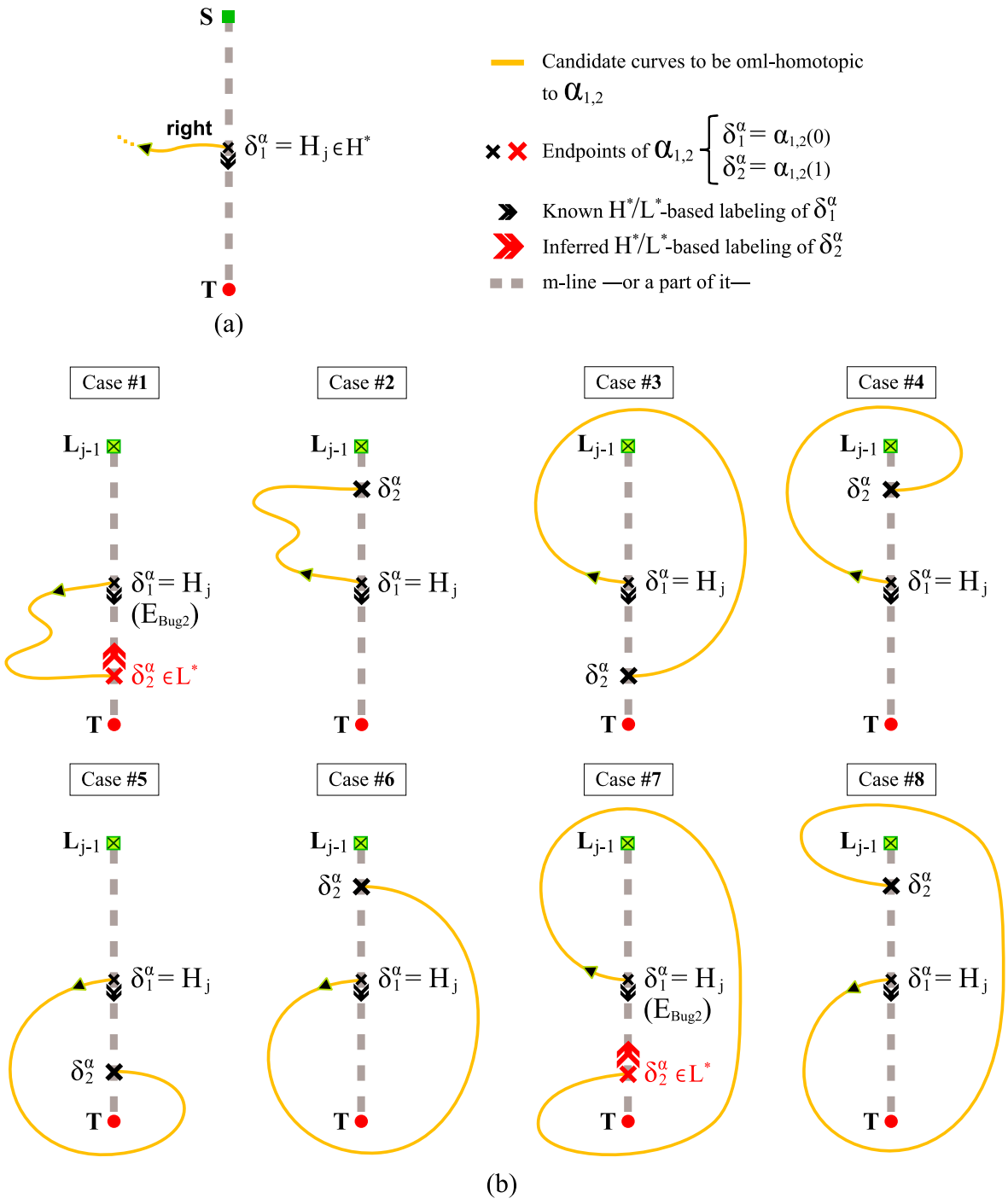


Figure 4.13: A coarse-filtered set of solutions for the shape of the primitive segment $\alpha_{1,2}$: (a) main filtering criterion; (b) ($L_{j-1}T$-adapted results.

it seems clear that $\alpha_{1,2}$ is not allowed to cross $(L_{j-1}H_j)^{mtg}$ at any point. Moreover, the position of $\delta_2^\alpha = \alpha_{1,2}(1)$ should be out of the segment ($L_{j-1}H_j^{mtg}$ —now, the expression includes H_j— since the endpoints of $\alpha_{1,2}$ should differ each other, i.e. $\delta_1^\alpha = \alpha_{1,2}(0) = H_j \neq \delta_2^\alpha$ (this is due to the fact that, according to definition 3, every boundary-following path / path segment defines a simple and open curve). Finally, by looking at cases 2, 4, 6, and 8 in figure 4.13(b), we rapidly conclude that they do not comply with the $\alpha_{1,2}$-based restriction revealed above.

- *About cases 3 and 5.* No line of reasoning leads to the rejection of cases 3 and 5 as possible $\alpha_{1,2}$'s shapes.

Summarizing, as a main outcome of the preceding study, the shape of the primitive segment $\alpha_{1,2}$ is guaranteed to be oml-homotopic to one curve or another of cases 3 and 5 in figure 4.13(b).

After exposing the two real possibilities for the topological appearance of $\alpha_{1,2}$, the shape-based analysis which has just been presented is going to be broaden to entirely cover the boundary-following path segment $\alpha_{1,3}$. Or, in other words, the shape of the proven-to-exist primitive segment $\alpha_{2,3}$ will be examined by keeping in mind every constraint (cstr) that follows:

cstr1. δ_2^α symbolizes the only point that is shared between the two consecutive primitive segments $\alpha_{1,2}$ and $\alpha_{2,3}$. Or from a simpler perspective, δ_2^α refers to a position where the m-line —ST— is crossed by $\alpha_{1,3}$. This double clarification of the meaning of δ_2^α allows us to formally affirm that: $\forall\epsilon > 0\ \exists i_0 \in (1-\epsilon, 1)$ *and* $i_1 \in (0, \epsilon)$ *such that* $\alpha_{1,2}(i_0)$ *and* $\alpha_{2,3}(i_1)$ *are ensured to be located in different sides of* ST. In essence, this statement provides a way for determining if $\alpha_{2,3}$ goes on either the left side or the right side of the m-line from δ_2^α. With this aim, let us exploit further the recently-gained knowledge respecting the feasible shapes of $\alpha_{1,2}$. On this matter, 3 and 5 are the two cases of interest for $\alpha_{1,2}$. In both cases, as can be verified in figure 4.13(b), those points of $\alpha_{1,2}$ situated in the neighborhood of δ_2^α lie on the left side of ST. Consequently, making use of the above-mentioned statement, exactly the opposite should occur with $\alpha_{2,3}$. Therefore, as a final result, $\alpha_{2,3}$ should extend towards the right side of ST from δ_2^α (see figure 4.14(a) for an at-a-glance justification of this fact).

At this moment, it is important to highlight that $\alpha_{1,2}$ was also resolved to extend, from its corresponding starting point —$\alpha_{1,2}(0) = \delta_1^\alpha$—, to the right side of the m-line like $\alpha_{2,3}$. The first constraint being supported takes advantage of such a coincidence by considering the shapes given in figure 4.13(b) as a complete set of topologically-different solutions for $\alpha_{2,3}$ —naturally, once the endpoints δ_1^α and δ_2^α have been replaced by δ_2^α and δ_3^α, respectively;

cstr2. No intersection is permitted between $\alpha_{1,2}$ and $\alpha_{2,3}$ since both primitive segments when taken as a whole, i.e. when being jointly represented by $\alpha_{1,3}$, are claimed to describe a simple and open curve;

cstr3. In view of lemmas 4 and 5, $\alpha(1)$ $\Big($ $= \alpha_{2,3}(1) = \delta_3^\alpha$, assuming that the boundary-following path α merely comprises two primitive segments$\Big)$ should be a point where the leaving arises for algorithm 4.1, but not for algorithm 4.2. Or, in more specific terms, the point linked to $\alpha(1)$ should involve both an L^*-type labeling —thus, $\alpha(1)$ will denote a potential leave point— as well as the exclusive fulfillment of condition C_1 from algorithm 4.1 —as opposed to condition C_3 from algorithm 4.2.

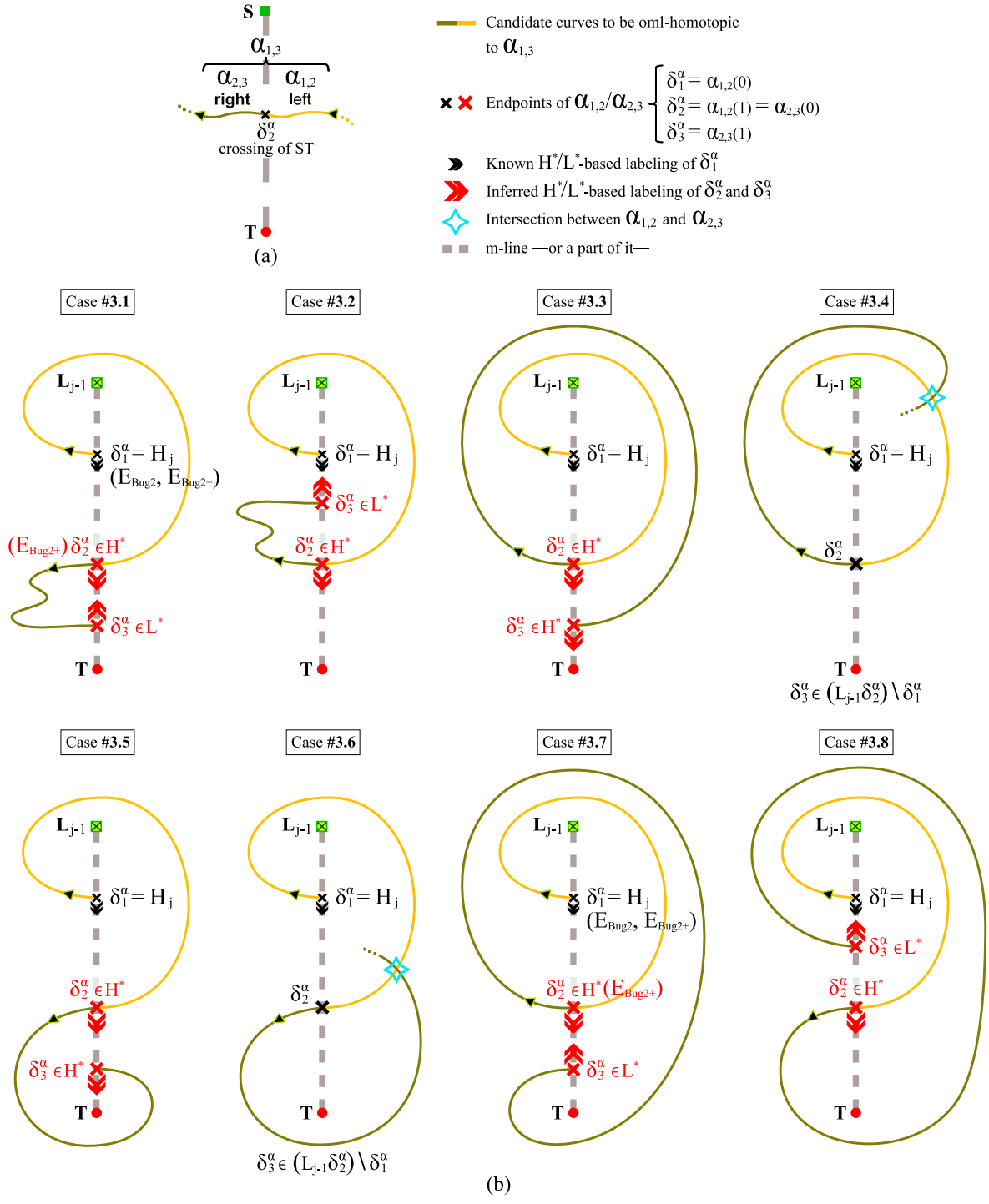


Figure 4.14: A first-round selection of shapes for the boundary-following path segment $\alpha_{1,3}$ on condition that $\alpha_{1,2}$ is the curve of case 3 in figure 4.13(b).

Figures 4.14(b) and 4.15(a) distinguish, from an oml-homotopic viewpoint, all possible shapes of the curve $\alpha_{1,3}$, which are obtained by combining cases 3 and 5 of $\alpha_{1,2}$ with each of the eight solutions for $\alpha_{2,3}$ that derive from the first constraint listed earlier $\Big($notice that, in the two referenced figures, a solution designated by $X.Y$ indicates that case X of $\alpha_{1,2}$ has been merged with case Y of $\alpha_{2,3}$; moreover, such a latter case ultimately corresponds to a version of case Y of $\alpha_{1,2}$ which, while keeping the distinctive topology of the original primitive segment, connects different endpoints —δ_1^α and δ_2^α versus δ_2^α and $\delta_3^\alpha$$\Big)$. Now, the sixteen resulting shapes for the boundary-following path segment $\alpha_{1,3}$ are checked against the remaining —second and third— constraints:

- *About cases 3.4, 3.6, 5.3, 5.4, 5.6, 5.7, and 5.8.* These cases reflect situations where the primitive segments $\alpha_{1,2}$ and $\alpha_{2,3}$ cross in, at least, one occasion $\Big($see figures 4.14(b) and 4.15(a) again; observe that, for the concerned cases, $\alpha_{2,3}$ is not always entirely drawn with the purpose of indicating that the curve extends, from the given dotted end, towards some point of the m-line segment ($L_{j-1}\delta_2^\alpha$) —not including $\delta_1^\alpha$$\Big)$. In this way, none of cases 3.4, 3.6, 5.3, 5.4, 5.6, 5.7, and 5.8 does satisfy our second mandatory constraint, which allows us to definitely get rid of them.

- *About cases 3.1, 3.7, and 5.1.* Figures 4.14(b) and 4.15(a) reveal the next three main facts regarding cases 3.1, 3.7, and 5.1: first of all, the inferred labeling of δ_3^α results to be of type L^*, i.e. δ_3^α is a potential leave point; in addition, on the basis of definition 2, the state of algorithms 4.1 and 4.2 are, respectively, $E_{Bug2} = d(\delta_1^\alpha, T) = d(H_j, T)$ and $E_{Bug2+} = d(\delta_2^\alpha$,T) at δ_3^α; finally, the expression $d(\delta_3^\alpha, T) < E_{Bug2+} < E_{Bug2}$ holds. In short, from above, we conclude that conditions C_1 and C_2 of algorithm 4.1, and conditions C_3 and C_4 of algorithm 4.2 will be certainly met when reaching the point δ_3^α once α ($= \alpha_{1,3}$) has been completely traversed. Nevertheless, this actually means that the leaving will take place for both algorithms at δ_3^α, and not only for algorithm 4.1 as required by our third constraint.

- *About cases 3.2, 3.8, and 5.2.* Lemma 4 describes α as a path that goes, in order, through the points H_j, H_k^*, and L_l^*. Besides, these points should comply with the inequality $d(H_k^*, T) < d(L_l^*, T) < d(H_j, T)$. As clearly shown in figures 4.14(b) and 4.15(a), cases 3.2, 3.8, and 5.2 fulfill the whole set of conditions previously mentioned. More exactly, the success of cases 3.2, 3.8, and 5.2 resides in becoming aware of the following: $\delta_1^\alpha = H_j$, δ_2^α is a potential hit point $\Big(= H_k^*\Big)$, δ_3^α symbolizes a potential leave point $\Big(= L_l^*\Big)$, and, at last, the relative position of such points on ST —specifically, on the m-line segment ($L_{j-1}T$— is given by $d(\delta_2^\alpha, T) < d(\delta_3^\alpha, T) < d(\delta_1^\alpha, T)$.

  After the preceding discussion, a verification of lemma 5 should be performed on each of the three solutions for the boundary-following path $\alpha = \alpha_{1,3}$ which have just been found. To this end, let us start by enumerating some of the properties of α that commonly apply to the cases at hand:

  1. $\Delta^\alpha = \{\delta_1^\alpha = H_j, \ \ldots \ , \delta_n^\alpha\}$ with $n = 3$;
  2. The boundary-following path segment $\alpha_{n-2,n} = \alpha_{1,3} = \alpha$ is oml-homotopic to that curve of figure 4.9(a) /(b) /(c) in case 5.2 /3.2 /3.8;
  3. As said before, $\delta_{n-2}^\alpha = \delta_1^\alpha = H_j \in H^*$, $\delta_{n-1}^\alpha = \delta_2^\alpha = H_k^*$, and $\delta_n^\alpha = \delta_3^\alpha = L_l^*$.

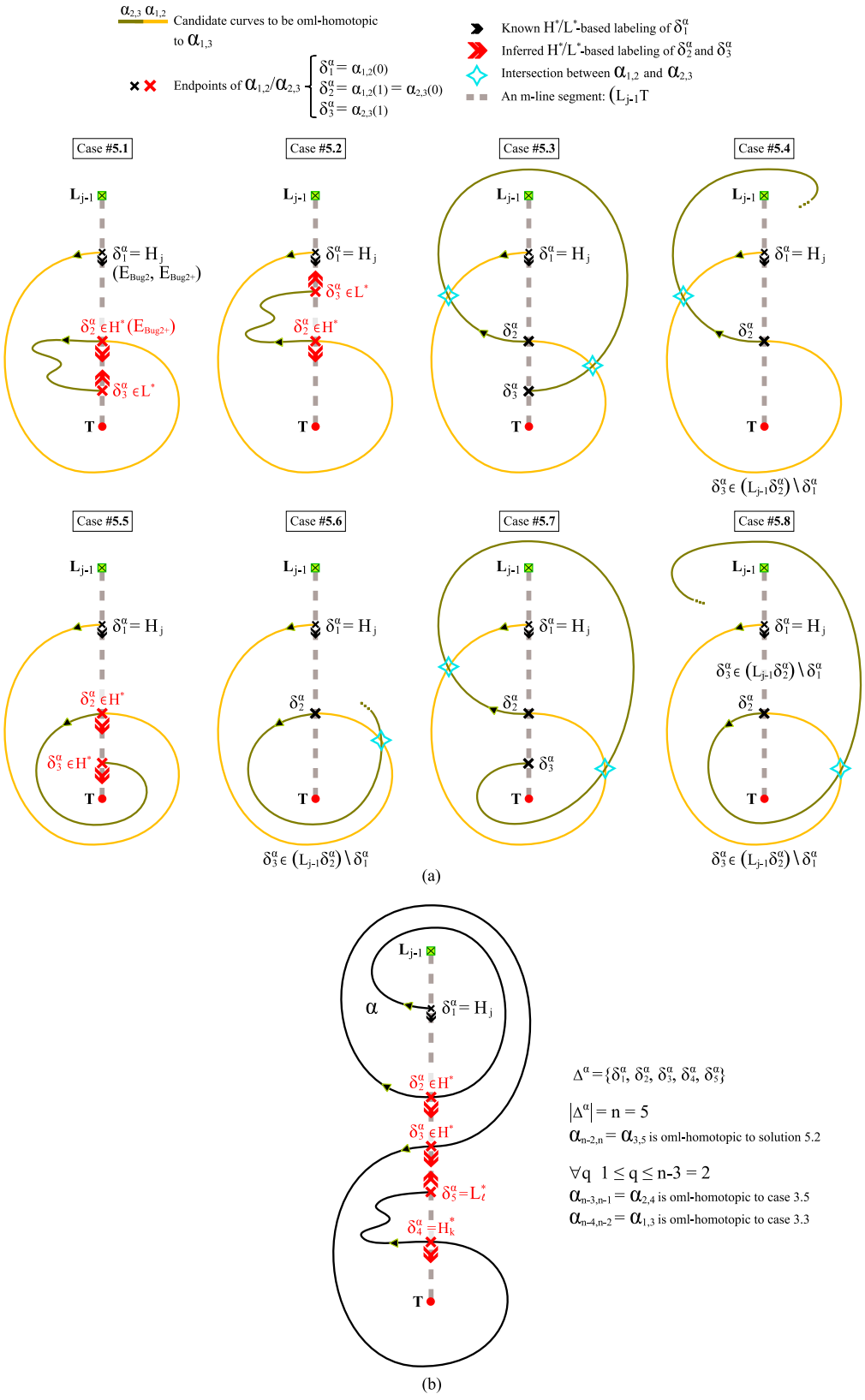


Figure 4.15: A first-round selection of shapes for the boundary-following path segment $\alpha_{1,3}$ on condition that $\alpha_{1,2}$ is the curve of case 5 in figure 4.13(b).

According to this, there is no doubt of a full agreement between the three aforelisted α's properties relating to cases 3.2, 3.8, and 5.2, and the ones stated by lemma 5.

- *About cases 3.3, 3.5, and 5.5.* Contrary to what has happened until now, no criterion can be used for either discarding or accepting cases 3.3, 3.5, and 5.5 as solutions to the boundary-following path α. This is essentially due to the fact that, in such cases, α ($= \alpha_{1,3}$) lacks for potential leave points, which makes sure to prevent algorithms 4.1 and 4.2 from abandoning the supposed contour following process initiated at $\delta_1^\alpha = H_j$. Consequently, under these circumstances, α will necessarily consist of an additional primitive segment —namely $\alpha_{3,4}$, being thus $\alpha = \alpha_{1,4}$—, which results from keeping on following the boundary of the obstacle once the point δ_3^α has been reached.

  Before going over the particular features of the new primitive segment $\alpha_{3,4}$, let us consider the aim of lemma 5. In a few words, this lemma tries to characterize the shape of a portion of the boundary-following path α. To be precise, some relevant information is provided by lemma 5 with respect to the boundary-following path segment $\alpha_{n-2,n}$ which, as can be guessed, comprises the two last primitive segments of α. Bearing this in mind, it seems clear that, in cases 3.3, 3.5, and 5.5, $\alpha_{2,4}$ constitutes the boundary-following path segment on which lemma 5 must be proved. Accordingly, $\alpha_{2,4}$ will become, hereafter, our main focus of attention, which actually means that, for the three currently analyzed cases, any other part of α —essentially, the primitive segment $\alpha_{1,2}$— will be completely ignored.

  Consistent with the above-mentioned purpose, notice, first of all, that the primitive segment $\alpha_{2,3}$ in case 3.3/cases 3.5 and 5.5 is oml-homotopically equivalent to $\alpha_{1,2}$ in case 3/case 5 (look at figures 4.13(b), 4.14(b), and 4.15(a) for appreciating such a correspondence). Hence, we can affirm that the formally-called shape space[12] linked to the primitive segment going after $\alpha_{2,3}$ in cases 3.3, 3.5, and 5.5 —i.e. the unknown $\alpha_{3,4}$— will definitely coincide with the one of the primitive segment following to $\alpha_{1,2}$ in cases 3 and 5. Or, in more general words, by accounting for both the given equivalence involving $\alpha_{2,3}$ and $\alpha_{1,2}$ as well as for the common shape space of their subsequent primitive segment, there is no doubt that no differences will exist between the corresponding shape spaces of the boundary-following path segments $\alpha_{2,4}$ and $\alpha_{1,3}$, as derived from cases 3.3, 3.5, and 5.5, and cases 3 and 5, respectively. In this way, remembering that figures 4.14(b) and 4.15(a) wholly illustrate the shape space of $\alpha_{1,3}$, we are now able to interpret these figures as also representing the shape space of $\alpha_{2,4}$ —obviously, with the condition that the points δ_1^α, δ_2^α, and δ_3^α are replaced by δ_2^α, δ_3^α, and δ_4^α, in that order.

  As significantly stated before, the shape space of the boundary-following path segment $\alpha_{2,4}$ is exactly the same as the one of $\alpha_{1,3}$, which includes the sixteen cases designated by 3.1, ..., 3.8, 5.1, ..., 5.8. All these cases —excepting the ongoing ones— have already been examined in search of solutions for $\alpha/\alpha_{n-2,n}$. Furthermore, each time a solution has been found, lemma 5 has been checked to be satisfied on it. In conclusion, when dealing with cases 3.3, 3.5, and 5.5, no new solutions for $\alpha_{n-2,n}$ will be obtained due to the recurrent analysis that certainly arises and leads, again and again, to the three previously identified solutions of cases 3.2, 3.8, and 5.2. Therefore, it is plain that cases 3.3, 3.5, and 5.5 are irrelevant under lemma 5.

[12] Generally speaking, the term *shape space* refers to the set of all possible shapes of a curve

Let us clarify the severe assertion finishing the last paragraph. It is important to know that cases 3.3, 3.5, and 5.5 are, in truth, influencing lemma 5 in the sense of considering α to be a boundary-following path that may consist of more than two primitive segments —if so, observe that $|\Delta^{\alpha}| = n > 3$. The value of n is only higher than 3 when cases/solutions 3.2, 3.8, or 5.2 are indirectly achieved from cases 3.3, 3.5, or 5.5. Concisely, the following properties hold for any α-compliant solution with $n > 3$: (1) $\alpha_{n-2,n}$ is oml-homotopic to one of the curves of cases 3.2, 3.8, and 5.2; (2) similarly, $\forall q\ 1 \leq q \leq n-3$, $\alpha_{n-2-q,n-q}$ is oml-homotopic to one of the curves of cases 3.3, 3.5, and 5.5; and, finally, (3) α defines a simple curve. By way of example, figure 4.15(b) sketches a solution for α having four primitive segments —i.e. $n = 5$.

Coming to an end, we have realized that cases 3.2, 3.8, and 5.2 represent the whole set of solutions for $\alpha_{n-2,n}$. As can be verified, these solutions faithfully correspond to the ones proposed by lemma 5 in figures 4.9(b), (c), and (a). Hence, lemma 5 is proved. □

Lemma 6. *Let α be the boundary-following path hypothesized in lemma 4 which is jointly traversed by algorithms 4.1 and 4.2 from $H_j = \alpha(0)$ to $L_l^* = \alpha(1)$ passing, in between, through H_k^* with $d(H_k^*, T) < d(L_l^*, T) < d(H_j, T)$. Then: in algorithm 4.1, once α has been completed and the leaving of such a contour following process has occurred at L_l^* ($= L_j$), H_k^* will become a real hit point ($\in H$) in some time before converging to T.*

Assumption/s. The same assumptions as in lemmas 4 and 5.

Proof. First of all, we insistently recommend the reader to revise every new concept that was introduced for proving lemma 5. Specifically, we are referring to: (1) on the one hand, the way of finding out the proper H^*/L^*-based labeling of the endpoints of a primitive boundary-following path segment; (2) on the other hand, the notation used for designating both the path resulting from the activation of the motion-to-goal behavior —XY^{mtg}—, and a fully/partially open straight-line segment —(XY)/(XY or XY).

Next, the proof of lemma 6 is widely developed on the assumption that algorithm 4.1 was configured with the parameter $pCFD = \mathit{right}$ when moving from H_j to L_l^*, i.e. when producing the boundary-following path named α (remember that such a configuration keeps constant throughout the execution of the algorithm). Under these circumstances, lemma 5 establishes that the shape space of the boundary-following path segment $\alpha_{n-2,n}$ is merely made up of the curves shown in figures 4.9(a), (b), and (c). This valuable knowledge in conjunction with the one issuing from lemma 4 that recognizes the interruption/abandonament of α at L_l^* in algorithm 4.1 are going to be used as a basis for the forthcoming $pCFD_{right}$-restricted proof.

Without loss of generality, the *right*-biased context which has just been set out for proving lemma 6 can be seen as equivalent to the one that stems from considering the horizontally-flipped version —around ($L_{j-1}T$— of each of the three aforesaid feasible $\alpha_{n-2,n}$'s shapes illustrated in figures 4.9(a,b,c), as long as any future operation of the contour following process /boundary-following behavior in algorithm 4.1 starts moving in the opposite direction to the suggested above —as a direct consequence of the turn—, meaning thus that $pCFD = \mathit{left}$. Figures 4.9(d), (e), and (f) reflect this different, but equivalent, context for the problem at hand which, as can be guessed, additionally corresponds to the one that would arise from a $pCFD_{left}$-restricted proof of the current lemma. Then, this fact allows us to affirm that the specific setting of the parameter $pCFD$ does not actually matter when trying to prove lemma 6, or in more convenient words, that the ongoing $pCFD_{right}$-based proof is also valid for the —neglected— case in which $pCFD = \mathit{left}$.

As briefly mentioned before, algorithm 4.1 stops running the boundary-following behavior that causes the generation of α when getting to the potential leave point L_l^*. At that moment, the other alternative behavior referred to as motion-to-goal takes control of the planner, and L_l^* turns into a real leave point with subscript j, i.e. $L_l^* = L_j$[13]. From this time forth, we are going to evaluate what may happen once the motion-to-goal behavior has been initiated at L_j in algorithm 4.1. This will be done by taking into account, one by one, all possible shapes of the boundary-following path segment $\alpha_{n-2,n}$. More exactly, to carry out the evaluation being proposed, we only need to consider the shape space of the primitive boundary-following path segment $\alpha_{n-1,n}$, and not that one of the whole curve $\alpha_{n-2,n}$. According to this and by observing figures 4.9(a), (b), and (c) again, one can rapidly realize that the shape space linked to $\alpha_{n-1,n}$ exclusively consists of the two curves appearing in figures 4.16(a) and (b).

Through the next lines, let us suppose the existence of a certain variable m initialized to zero —$m = 0$—, as well as that the shape of $\alpha_{n-1,n}$ is oml-homotopic to the curve of figure 4.16(a). Under these circumstances, the list of all relevant events that may be triggered from $L_j = L_{j+m}$ in algorithm 4.1 is as follows:

e1. As seems clear, in case that no obstacles are crossing the straight-line segment $(L_{j+m}H_k^*)$, the planner will necessarily reach H_k^* as a natural result of the activation of the motion-to-goal behavior at L_{j+m}.

e2. Unlike the preceding situation, let H_{j+1+m} be the point where an obstacle has been found before reaching H_k^*, while performing the motion-to-goal behavior invoked at L_{j+m}. Seeing that, algorithm 4.1, after defining such a real hit point H_{j+1+m} and setting $E_{Bug2} = d(H_{j+1+m}, T)$, will immediately start traversing the contour of the new detected obstacle in right direction, as given by $pCFD = right$. Or from a different perspective, algorithm 4.1 will generate a new boundary-following path —named β^m, hereafter—, which will extend on the right side of $(L_{j-1}T$ from H_{j+1+m}[14]. Accordingly, let $\beta_{1,2}^m$ denote the first primitive boundary-following path segment of β^m. Notice that the endpoints of this primitive segment are characterized by $\beta_{1,2}^m(0) = \delta_1^{\beta^m} = H_{j+1+m}$ and $\beta_{1,2}^m(1) = \delta_2^{\beta^m}$, being the latter certainly located in some position along ST based on definitions 3 and 4. Now, prior to identifying the shape space of $\beta_{1,2}^m$, we should pay special attention to a couple of facts. On the one hand, $\forall q\ 0 \leq q \leq m$, $(L_{j+q}H_{j+1+q})^{mtg}$ designates a path that is ensured to be free of obstacles because an earlier execution of the motion-to-goal behavior in algorithm 4.1 successfully accomplished to get across it. On the other hand, by realizing —in line with the *Jordan curve theorem*— that obstacles do not touch each other, we can deduce that $\beta_{1,2}^m$ is intersecting neither $\alpha_{n-1,n}$ nor $\beta_{1,2}^q$ $\forall q\ 0 \leq q < m$[15]. In short, by putting the previous facts together, it seems obvious that there is no other option for the endpoint $\delta_2^{\beta^m}$ apart from lying on the m-line segment $(H_{j+1+m}H_k^*)$. Furthermore, this result, added to the knowledge that any primitive segment —therefore, $\beta_{1,2}^m$— describes a simple curve, yields to the final conclusion that the shape space of $\beta_{1,2}^m$ contains only the curve depicted in figure 4.16(c) (see also figure 4.16(d) where a representative set of unfeasible shapes for $\beta_{1,2}^m$ is provided). Or using more formal terms, we can equivalently state that every feasible shape for the

[13] The rule guiding the assignation of a subscript to a new L-type point simply says that such a subscript should agree with the one given to the most recently-defined real hit point. In particular, this last H-type point corresponds to $H_j = \alpha(0)$ when algorithm 4.1 is located at $L_l^* = \alpha(1)$

[14] Look from X to Y for suitably determining the left / right side of a generic straight-line segment XY

[15] If $m = 0$, the expression does not involve any curve $\beta_{1,2}^q$

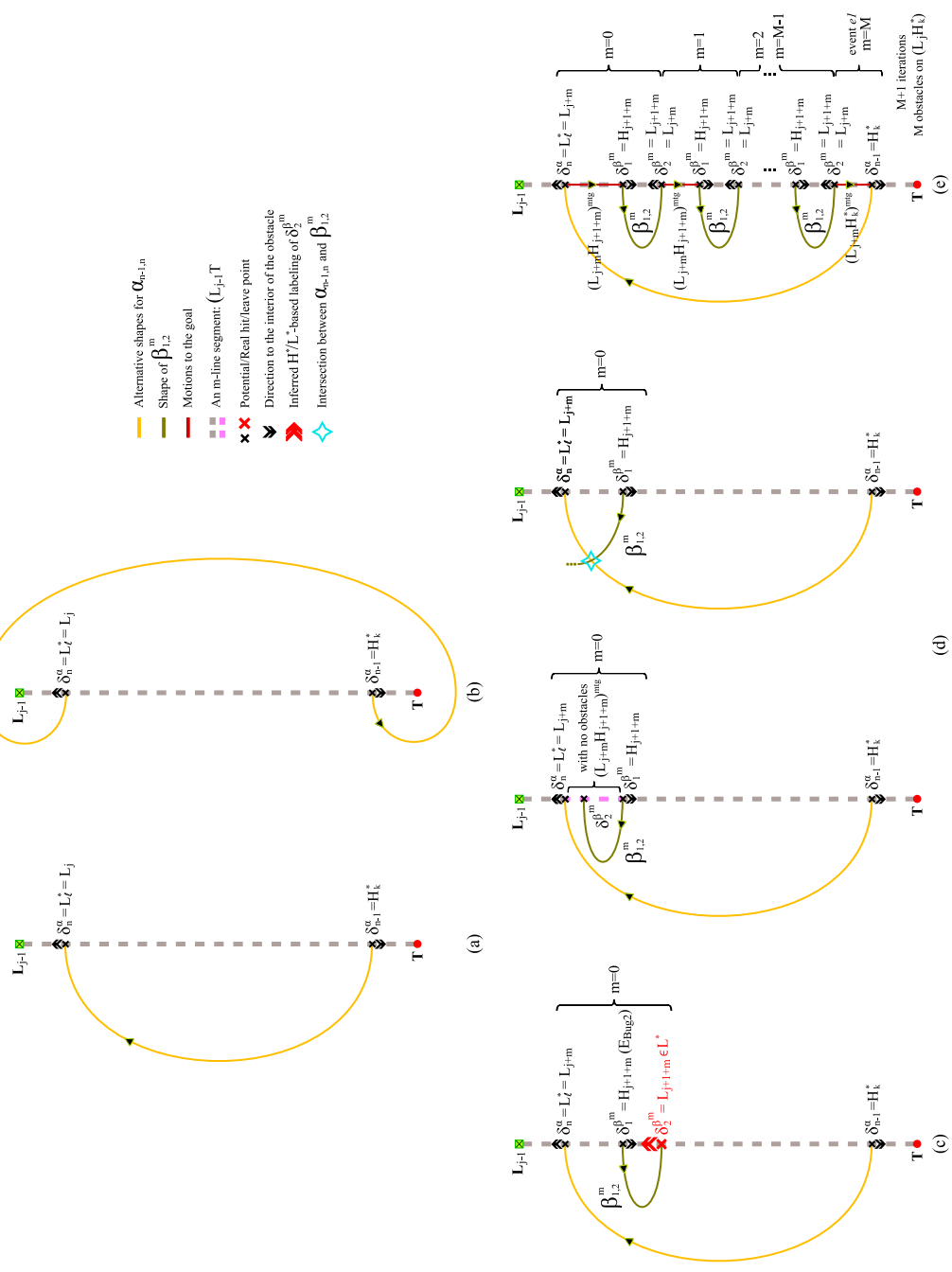


Figure 4.16: (a,b) all possible shapes of $\alpha_{n-1,n}$; (c,d,e) illustrating what can and cannot happen after algorithm 4.1 has initiated the motion-to-goal behavior at L_j.

primitive segment $\beta_{1,2}^m$ is oml-homotopic to the curve of figure 4.16(c). Algorithm 4.1 will traverse $\beta_{1,2}^m$ from $\delta_1^{\beta^m} = H_{j+1+m}$ to $\delta_2^{\beta^m}$. Nevertheless, once the point $\delta_2^{\beta^m}$ is reached, the ongoing contour following process will be replaced by the motion-to-goal behavior —the same that happened when moving to L_{j+q}, $\forall q\ 0 \leq q \leq m$— since the two conditions controlling the indicated transition will be met: firstly, and as it is illustrated in figure 4.16(c), $\delta_2^{\beta^m}$ is known to be a potential leave point —$\in L^*$— in accordance with the so-called labeling inference method explained during the proof of lemma 5; secondly, $d\Big(\delta_2^{\beta^m}, T\Big) < E_{Bug2} = d(H_{j+1+m}, T)$.

Following the discussion above, observe that $\delta_2^{\beta^m}$ will become the real leave point L_{j+1+m} as a result of the last activation of the motion-to-goal behavior. Thus, our concern now is about what may happen from L_{j+1+m} according to algorithm 4.1. With this aim, recall that the variable m is assumed to be zero when referring to L_{j+1+m}. Consequently, if m is incremented by one, such a point can be alternatively denoted as L_{j+m}. Let m be 1. Then, events *e1* and *e2* reflect, once again, the whole set of situations to which algorithm 4.1 could be confronted after defining $L_{j+1} = L_{j+m}$.

Summarizing, the given description of event *e2* is presented as an iterative process, where the number of iterations to be performed —each of them operating with an increased value of m that starts from zero— directly depends on the amount of obstacles that algorithm 4.1 will find on $(L_j H_k^*)$. It is important to remark here that this number of iterations is guaranteed to be finite because the set of all obstacles in the environment is also finite by hypothesis, and because, at the same time, the circumnavigation of those obstacles detected on $(L_j H_k^*)$ does always mean a step forward towards H_k^*, which clearly discards that an obstacle that has already been overcome can be later revisited. As can be guessed from the preceding words, algorithm 4.1, in each iteration, achieves progress on its way from L_j to H_k^*. Moreover, the end of the iterative process arises when no more obstacles are impeding such a progress, implying thus the triggering of event *e1* and, what is more interesting for us, the convergence of algorithm 4.1 to H_k^*. To finish, figure 4.16(e) shows how the suggested *e2*-derivative iteration process works from a generic viewpoint.

□

Theorem 2. *The length of the path produced by algorithm 4.2 is, at worst, equal to the length of the path generated by algorithm 4.1.*

Assumption/s. Algorithms 4.1 and 4.2 are executed on the same scenario. Additionally, a path exists between S and T, and the parameter $pCFD$ is set to the same value in both strategies.

Proof. The forthcoming proof relies on the general idea that algorithm 4.2 generates exactly the same path as algorithm 4.1 in all cases, except in those ones where cycles appear; in these special cases, algorithm 4.2 does improve the path given by algorithm 4.1 by reducing the number of resulting cycles, as it is evidenced in figure 4.17 (observe how the red-marked cycle produced by algorithm 4.1 disappears when executing algorithm 4.2). In short, the above allows us to alternatively imagine algorithm 4.2 as a method for removing some cycles —if any— from a path derived from algorithm 4.1, while keeping unaltered the rest of the trajectory. In view of that, it seems clear that algorithm 4.2 will outperform algorithm 4.1

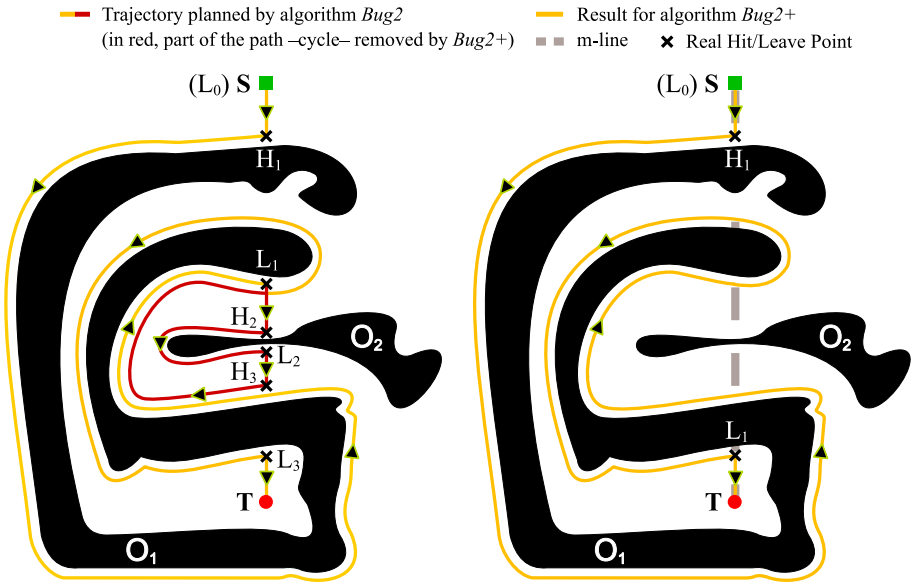


Figure 4.17: Understanding algorithm 4.2 as a method for removing cycles from a path planned by algorithm 4.1. These results were obtained by configuring the two strategies to follow the boundary of the obstacles towards the right ($pCFD = right$).

when being able to successfully carry out —partially or totally— the alluded removal of cycles, whereas, otherwise, both algorithms will offer an identical path length performance. This concluding remark is formally treated and used next as a guide for proving theorem 2.

After jointly starting at S, algorithms 4.1 and 4.2 are known to behave equivalently while the circumstances described in lemma 4 do not arise. Bearing this in mind, there is not doubt that these algorithms will provide the same solution to the path-planning problem if lemma 4 never applies. Let us now consider the non-trivial case in which algorithm 4.1 and algorithm 4.2 converge to T in a different way. So, as pointed out earlier, this requires working in the context of lemma 4, which means that both algorithms are going to cross the m-line at points H_k^* and L_l^* during a supposed activation of the contour following process initiated at H_j. Furthermore, the crossing linked to H_k^* should occur before the one of L_l^*, and the inequality $d(H_k^*, T) < d(L_l^*, T) < d(H_j, T)$ should hold as well. On the basis of the concept of *oml-homotopy* (see definition 4 for details concerning this adapted topological concept), lemma 5 characterizes the shape of the above-mentioned boundary-following path. This path is generically referred to as α hereafter. Figure 4.18(a) shows the results of such a characterization, but mainly focusing on the part of α represented by $\alpha_{n-1,n}$. Specifically, $\alpha_{n-1,n}$ denotes the last primitive segment of α, whose endpoints are $\alpha_{n-1,n}(0) = \delta_{n-1}^{\alpha} = H_k^*$ and $\alpha_{n-1,n}(1) = \delta_n^{\alpha} = L_l^*$.

As already discussed, algorithms 4.1 and 4.2 traverse α together until getting to L_l^*. From that moment, however, their paths diverge as stated by lemma 4. More precisely, no new decision is taken by algorithm 4.2 at point L_l^*, continuing thus its ongoing contour following process —i.e. extending α. On the contrary, algorithm 4.1 decides to abandon the boundary of the current obstacle to try to progress towards the target by moving along L_l^*T. To this respect, lemma 6 gives some clues about the progress that is made by algorithm 4.1 after leaving L_l^*. In short, and as can be observed in figure 4.18(b), algorithm 4.1 surely arrives at H_k^* —let us highlight that this will be the second time in achieving the potential hit

point H_k^* since $\alpha_{n-1,n}(0) = H_k^*$. Lemma 6 also establishes that algorithm 4.1 defines a new H-type point when returning to H_k^*. This is because such an algorithm detects, at H_k^*, an apparently new obstacle[16] impeding further progress to T on the m-line. It is important to note that there are two more consequences of hitting an obstacle at H_k^*. On the one hand, the state of algorithm 4.1 changes to $E_{Bug2} = d(H_k^*, T)$. On the other hand, the generation of an additional boundary-following path named γ starts with the intention of circumnavigating the obstacle which has been found. Then, according to the latter fact, let $\gamma_{1,2}$ be the first primitive segment of γ such that $\gamma_{1,2}(0) = H_k^*$. As it seems obvious, to properly determine now the form of $\gamma_{1,2}$, we need some information on the precise shape of the contour of the detected obstacle, essentially nearby H_k^*. Fortunately, this information can be actually gained by realizing all of the following: (1) in line with definition 3, any boundary-following path ultimately corresponds to a portion of the contour of a certain obstacle; (2) the boundary-following path α passes over H_k^* —$\alpha_{n-2,n-1}(1) = \alpha_{n-1,n}(0) = H_k^*$— just like γ, which clearly indicates that α and γ result from traversing the contour of the same obstacle. In conclusion, with the above evidences in mind, we can affirm that the form of $\gamma_{1,2}$ necessarily coincides with the one of the primitive segment either $\alpha_{n-2,n-1}$ or $\alpha_{n-1,n}$. Moreover, the deciding factor between these two choices depends on the direction taken by algorithm 4.1 for following the contour of the obstacle from $\gamma_{1,2}(0)$. Remember that this direction —in broad terms, *left* or *right*— is resolved by using the parameter $pCFD$. As a final point, figure 4.18(c) definitely demonstrates that $\gamma_{1,2} = \alpha_{n-1,n}$ by considering the value of $pCFD$ intrinsically associated with each possible shape of α. Therefore, after completing the primitive segment $\gamma_{1,2}$, algorithm 4.1 ends up at L_l^* in view of $\gamma_{1,2}(1) = \alpha_{n-1,n}(1) = L_l^*$. □

Corollary 1. *The length of a path planned by algorithm 4.2 is bounded by expression 4.1.*

Proof. Theorem 2 states that algorithm 4.2 never provides a path longer than algorithm 4.1. On the other hand, as it is well-known (see [53]), no path is given by algorithm 4.1 whose length exceeds the limit specified by expression 4.1. Consequently, such an upper bound is also valid for algorithm 4.2, proving thus corollary 1. □

[16] Neither algorithm 4.1 nor algorithm 4.2 is able to distinguish whether an obstacle has already been visited or not

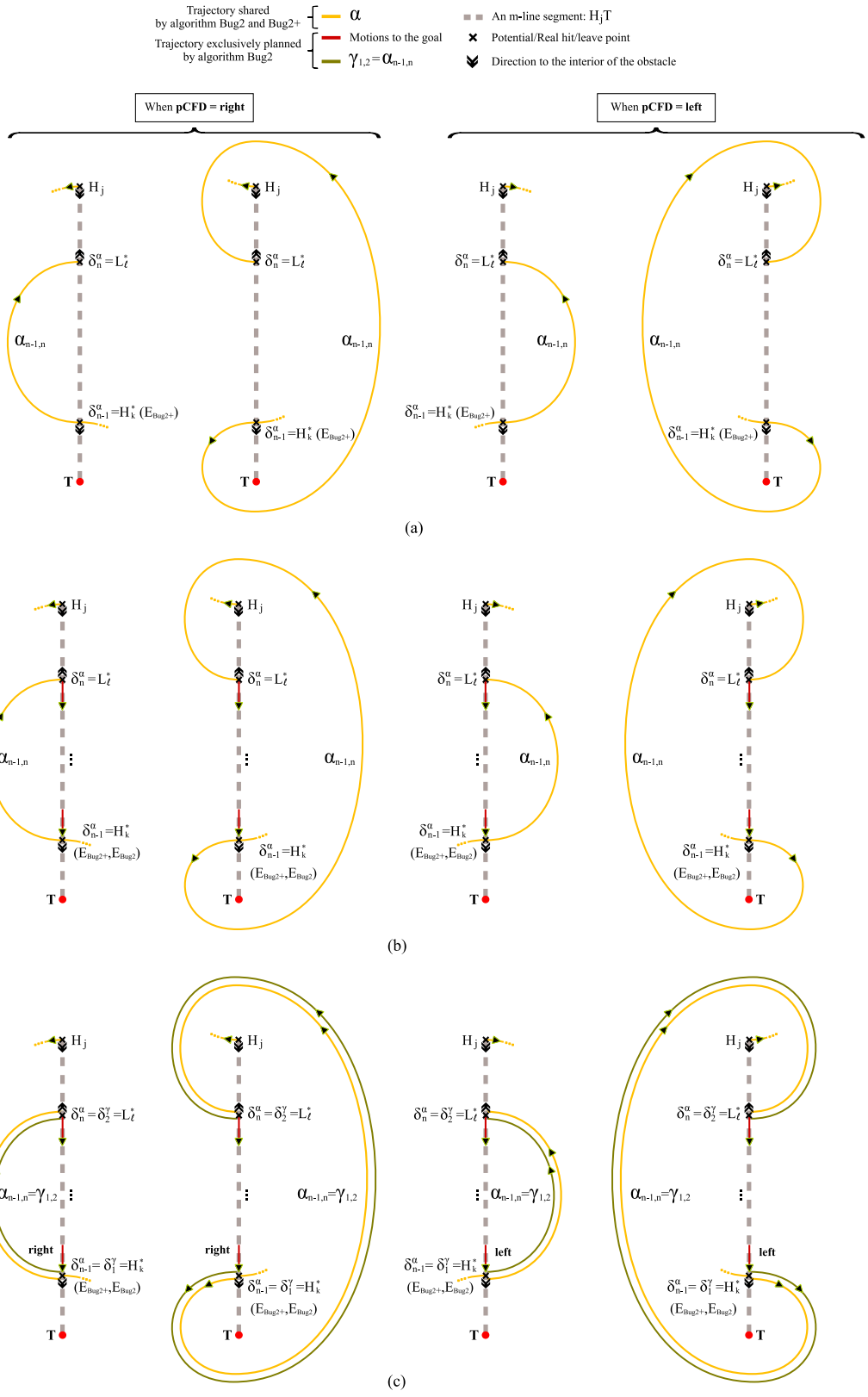


Figure 4.18: Analyzing the way in which the paths produced by algorithms 4.1 and 4.2 can differ from each other.

T^2-based Reactive Navigation with Global Proofs

Any *purely* reactive navigation method, and, in particular, any strategy built upon the concepts of *Traversability and Tenacity (T^2)* —refer to chapter 3—, is not able to always ensure convergence to the target position. This shortcoming appears due to the well-known fact that these strategies make their decisions based solely on the (local) information about the environment that is currently / has recently[1] been supplied by the robot's sensors.

Alternatively, as compared to the above class of techniques, reactive methods of *non-pure* type generally offer a richer way of navigation by removing the requirement for the exclusive use of local environmental information when planning the robot motion. To this respect, nevertheless, it is appropriate to clarify that only a limited amount of global information can be really managed by these methods —of the order of a few bytes— and, besides, an extremely simple processing of this information should be carried out so as to avoid losing reactiveness. As a close example, the algorithm named *Bug2+*, which was described in chapter 4, adopts such a non-pure reactive paradigm. Specifically, this algorithm exploits the open possibility of dealing with global information to guarantee the completion of the navigation task —whenever a solution exists. In addition, notice that, as inherently demanded by its own paradigm, the accomplishment of the property of completeness in Bug2+ just involves little reasoning on a small set of data; to be more precise, convergence arises as the result of taking local decisions that mainly focus on the reduction of a certain distance (D) —D is the whole set of data to be considered on a global level—, which represents the nearest the robot has been, until that moment, to the desired target from the m-line[2].

As has previously been pointed out, non-pure methods of reactive navigation do combine non-computationally-intensive local decisions with very limited global information to typically attain some kind of completeness —or in other words, to move far beyond the complexity of the tasks that are affordable by the purely reactive paradigm. As a general remark, it is important to note that completeness necessarily requires global information. What is more, the variety of broadly different ways in which a method that is intended to be complete could work becomes less diverse as the amount of global information that the method is allowed to collect and employ decreases. Keeping the last observation in mind, it seems perfectly reasonable the fact that all non-pure reactive navigation methods exhibit the same overall pattern of behavior to gain completeness. As one can easily deduce, such a pattern

[1] In practical terms, as actually discussed in section 3.1.2.3, a purely reactive robot is allowed to possess a memory to keep track of all sensor data collected over a short period of time; although far from obvious, the use of this memory provides the robot with a better / not-so-local understanding of its surroundings, with the ultimate aim of increasing the global efficiency of the robot actions as well as the chances to reach the target

[2] As defined in section 4.1.2, this is the line segment whose endpoints are the start and target locations of the intended navigation task

is fully influenced by the minimal global understanding of the navigation environment that these methods can get to have by construction. Going briefly into the details of this pattern, it essentially consists of the following traits: (1) any non-pure reactive navigation method restricts the motion of the robot to a small set of predefined actions such as, for instance, moving straight towards the target location or moving around a specific obstacle; (2) just one action is permitted to be performed by the robot at a time; (3) a condition is defined to trigger the execution of each action (observe that, as stipulated by point 2, a new triggered action definitely replaces the previous one); (4) the conditions of point 3 are chosen in such a form that the simultaneous triggering of several actions cannot occur; (5) once again regarding the conditions of point 3, it should be finally stressed that they used to be quite conservative, in the strict sense that a condition is only satisfied when the taking of its associated action is going to surely imply a step forward in the successful achievement of the navigation task. (An example of the latter is found in the algorithm Bug2+: first of all, and as widely explained in section 4.2, this algorithm contemplates two actions[3] named motion-to-goal and boundary-following; in relation to point 5, the conservative character of Bug2+ becomes absolutely clear under the motion-to-goal action, since it is applied exclusively when there is a total certainty that, by moving directly towards the target, the so-called distance D will be reduced; as an additional comment, notice that, to a general extent, the distance D indicates the progress that has been made to the target —the less is this distance, the greater is such progress, and, therefore, the more are the Bug2+'s chances for promptly completing the navigation task.)

Now, before detailing the work presented in this chapter, let us rather examine the most relevant implications that are inherently associated with point 5 of the above pattern of behavior. First of all, it is patently obvious that being conservative in the specification of the action-triggering conditions does help a non-pure reactive navigation method to ensure completeness / global convergence —recall that, when actions are treated conservatively, they take place upon a basis of continuous global progress to the target. Unfortunately, point 5 also brings a major disadvantage to the non-pure reactive paradigm, because of negatively affecting its path length performance. This disadvantage derives fundamentally from the fact that there is a late identification —and, in consequence, execution— of the henceforth named proper action (concisely, the term 'proper action' refers here to the action whose execution would currently entail a global advance in reaching the target). As generally indicated by point 5, non-pure reactive navigation methods identify the proper action by means of the action-triggering conditions, following a predominantly conservative approach. This approach particularly demands that the identification of the proper action is highly —or even 100-percent— reliable. As seems evident, to gain such a high level of reliability, there is a need for certain precise global knowledge of the navigation environment (specifically, this necessity becomes absolutely clear when explicitly highlighting what is behind a reliable identification of the proper action; in short, it requires to accurately anticipate the effects that each of the possible actions defined at point 1 would presently have on the progress of the robot towards the desired target within the given navigation environment; as the final part of this identification, the action certainly expecting to have positive effects on the robot's progress is chosen for being immediately carried out —exceptionally, notice that, in case that no action appears to be definitely favoring the robot's progress, the identification is considered to fail, and the execution of the action which was last successfully identified as proper is maintained). At this moment, it is important to remember that all non-pure reactive navigation methods

[3]In the context of the algorithm Bug2+ —and of any other Bug-like strategy—, actions are referred to as primitive behaviors (or, simply, behaviors)

are merely allowed to allocate very scarce memory resources for the accumulation of global information about the environment. This limitation justifies that these methods suffer, on many occasions while performing the navigation task, a mismatch between the real and their global understanding of the environment. Moreover, in these methods, due to the adoption of a conservative approach, each occurrence of the aforesaid mismatch ultimately means a failure in the identification of the proper action, or in other words, the incapacity for knowing which is the action that best fits the current circumstances of navigation. In closing, the preceding observations explain why, in any non-pure reactive navigation method, the identification —and execution— of an action as proper, habitually happens later than the time in which this action began to be truly proper. This delay in suitably adapting the robot's action to the immediate navigation needs does inevitably lead to suboptimal final trajectories. To further support this conclusion, figure 5.1(a) depicts, and deeply analyzes using the previously introduced concept of proper action, the far-to-optimal trajectory generated by our non-pure reactive navigation algorithm Bug2+ in a simple scenario with an only obstacle having a slightly deformed U-type shape. Additionally, this same figure, in conjunction with figure 5.1(b), brings to light the general fact that the path length performance of a non-pure reactive navigation method may be worse than the one of a method built under the (more restrictive) purely reactive paradigm —with this mainly occurring in scenarios composed of non-intricate obstacles. Furthermore, with specific regard to the algorithm Bug2+ and the strategies based on the T^2 concepts, figures 5.1(a) and (b) make evident what was amply demonstrated in the study of section 3.2.4.2 when experimentally evaluating both approaches[4]. From this study, it was concluded that, in environments where a T^2-based strategy successfully makes the vehicle reach the target, this strategy does provide —in most of the cases— significantly shorter resultant trajectories than the algorithm Bug2+.

In this chapter, we present a new method of reactive navigation named *Bug*T^2, which arises as a combination of the algorithm Bug2+ and the T^2 navigation framework. With BugT^2, we are intended to make the most of the two techniques being combined, which means that BugT^2 should be able to: (1) prove global convergence to the target destination —when it is reachable—, just as done by Bug2+; and (2) follow a path towards such a target similar in length to the one that would be produced in accordance with the T^2 principles. Completing the general outline of the proposal, it is worth to mention that BugT^2 is a *non-pure* method that enables a robot to reactively navigate through planar environments populated by unknown stationary obstacles.

The remainder of this chapter is organized as follows: first of all, section 5.1 includes a few definitions and establishes the necessary notation for the subsequent sections; after that, a geometrical description of both the algorithm Bug2+ and the T^2 navigation framework is given in section 5.2; then, section 5.3 uses the descriptions of section 5.2 as guidelines for devising the novel non-pure reactive navigation method called *Bug*T^2; and, finally, section 5.4 verifies that BugT^2 does really offer the best of the Bug2+ and T^2 approaches.

[4] To be honest, the study of section 3.2.4.2 did conduct experiments with the algorithm Bug2, and not with the improved version of this algorithm, called Bug2+, that was put forward in chapter 4. Nevertheless, the reader should note that the same results would have been obtained by applying either Bug2 or Bug2+ on the set of scenarios that such a study used as test bed (see figure 3.18 for details about the characteristics of all these scenarios). With this in mind, it seems plainly clear that the conclusions that the study drew from comparing the results of the algorithm Bug2 with those achieved through a T^2-based navigation —namely, under the operation of the strategy *Random* T^2— are perfectly valid for describing the relative performance between the algorithm Bug2+ and the T^2 navigation framework

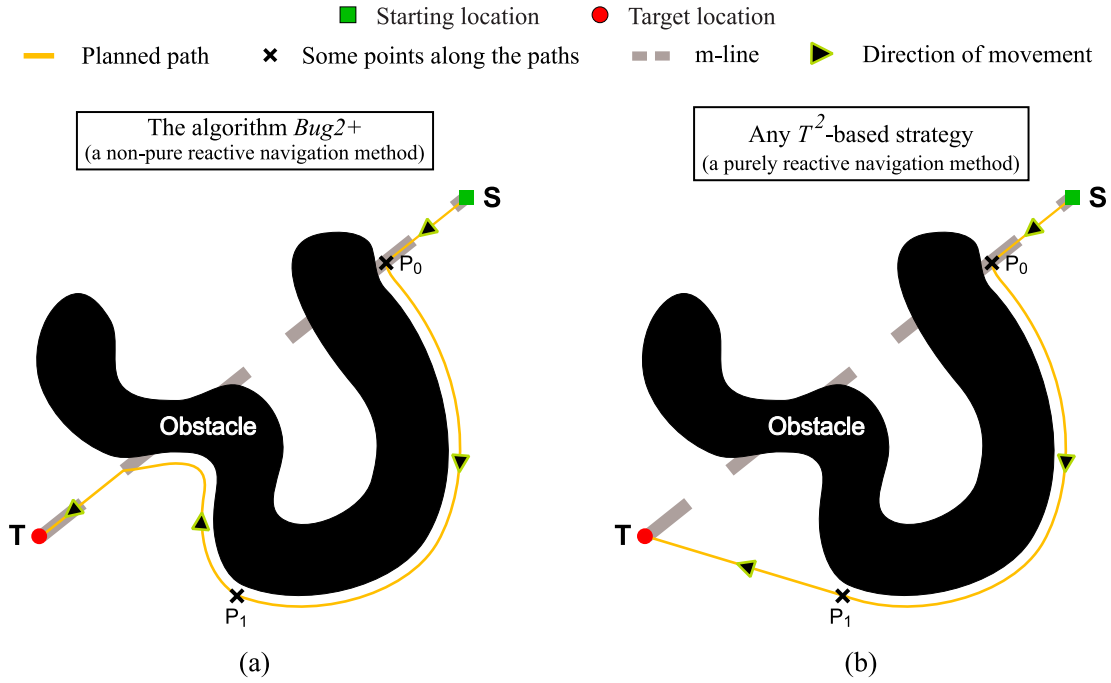


Figure 5.1: Comparison in terms of path length performance between the algorithm Bug2+ and any T^2-based strategy, assuming that both methods decide at point P_0 to circumnavigate the contour of the obstacle to the left. The forthcoming discussion just focuses on the moment in which the trajectories produced by these methods begin to differ, i.e. at point P_1. In (a), the boundary-following action initiated at point P_0 stops being really *proper* at point P_1, since, from there, it is plainly better to move straight towards the target (T) by invoking the alternative go-to-goal action; however, as can be seen, the algorithm Bug2+ *conservatively* discards taking the go-to-goal action at P_1 —meaning thus that the boundary-following action remains in execution—, because of not having enough certainty that the go-to-goal action is going to make some progress in meeting T $\Big($to be more precise, the underlying reasons for such a discarding decision are revealed in the following: first of all, notice that (1) Bug2+ utilizes the so-denoted distance D as its progress-to-target measure; besides, in Bug2+, (2) an action is performed solely when knowing that it will doubtlessly lead to reducing D; as a last general remark about Bug2+, it should be also stressed that (3) a reduction of the distance D involves the achievement of a point on the m-line closer to T than all other m-line's points previously encountered; now looking, in particular, at case (a) of this figure, it seems clear from remarks 1 to 3 that (4) the taking of the go-to-goal action at P_1 requires an absolute guarantee that this action will cause the reaching of a m-line's point nearer T than P_0 —observe that, when the robot is located at P_1, P_0 designates the earlier-visited m-line's point which is the closest to T; unfortunately, (5) it is not possible to guarantee the above because Bug2+ has no information concerning the obstacles that could find by moving straight from P_1 to $T\Big)$. In (b), the T^2-based strategy behaves exactly like Bug2+ until arriving at P_1; at this point, the movement along the contour of the obstacle is abandoned to try to attain T by following a straight-line path; more specifically, the T^2-based strategy recognizes P_1 as a point from where reaching T could be feasible by means of a direct path; as becomes obvious, its feasibility depends on the absence, or not, of obstacles in the unexplored region of the environment defined by the straight-line segment with endpoints P_1 and T; at P_1, the T^2-based strategy does not know if there exists an obstacle-free path directly joining P_1 and T, but, even so, it prefers to take the risk of moving towards T.

5.1 Definitions and Notation

5.1.1 Definitions

Some concepts related to planar curves are reviewed here.

D1. A parameterized planar curve is a continuous mapping $\gamma : [0,1] \rightarrow \mathbb{R}^2$. In addition, γ is said to be:

simple if $\gamma(t_i) = \gamma(t_j)$ only when either $t_i = t_j$ or $\{t_i, t_j\} = \{0,1\}$.

closed if $\gamma(0) = \gamma(1)$.

Jordan if it is both simple and closed. As intuitively obvious, a Jordan curve splits the plane $\mathbb{R}^2$ into two distinct connected regions having the curve as a common boundary. One of these regions is bounded —the interior—, while the other is unbounded —the exterior.

regular if the first derivative $\gamma'(t)$ exists and is non-zero for any $t \in \ ]0,1[$.

oriented if the set $\{\gamma(t) \mid t \in [0,1]\}$ —or, equivalently, the image set of γ— is ordered in terms of increasing t. More informally speaking, an oriented curve defines a direction of travel along the curve; specifically, this direction corresponds to the one going from $\gamma(0)$ —the starting point— to $\gamma(1)$ —the ending point.

Figure 5.2(a) provides an example of each of the types of planar curves early explained.

D2. A *supporting line* of an oriented planar curve γ at a point $\gamma(t_i)$ with $t_i \in \ ]0,1[$ is an oriented line, $Sl^{\gamma}_{t_i}$, that satisfies: (1) $Sl^{\gamma}_{t_i}$ coincides with the line tangent to γ at the point $\gamma(t_i)$; besides, (2) $Sl^{\gamma}_{t_i}$ points in the direction of travel of γ at $\gamma(t_i)$; and, lastly, (3) $Sl^{\gamma}_{t_i}$ leaves the whole curve γ in one of the two closed half-planes in which $Sl^{\gamma}_{t_i}$ divides $\mathbb{R}^2$ $\Big($as needed for a forthcoming definition, the two closed half-planes delimited by $Sl^{\gamma}_{t_i}$ are distinguished by means of the labels 'left' and 'right'; the left —alternatively, right— closed half-plane is the one situated on the left —alternatively, right— hand side of $\gamma(t_i)$ when looking in the direction $Sl^{\gamma}_{t_i}$ points to$\Big)$.

Figure 5.2(b) illustrates the concept of supporting line using the oriented-type curve of figure 5.2(a)[5].

D3. An oriented planar curve γ is referred to as *convex* if $\forall t \in \ ]0,1[$, (1) Sl^{γ}_{t} exists and (2) γ completely lies in the right-labeled closed half-plane of Sl^{γ}_{t}.

By way of clarification, we next analyze the three oriented-type curves of figure 5.2(c) from the viewpoint of convexity by taking into account the proposed definition. Moving from left to right across the figure, the first curve is not convex because there exists at least a non-extreme point of the curve that does not have a supporting line —hence, condition 1 is not fulfilled; likewise, the second curve is not convex either, but now due to the fact that there exists at least a non-extreme point of the curve whose supporting line

[5] Notice that such a curve, in addition to being oriented, is also regular. In the context of definition D2, regularity is a desirable feature due to the following: as can be guessed from condition 1, a supporting line does require the existence of the line tangent to the given curve γ at the specified point t_i; regular planar curves inherently meet this requirement, since these curves do not have singular points such as, for instance, cusps —see the right-most plot of figure 5.2(a) for an example of a curve with singularities

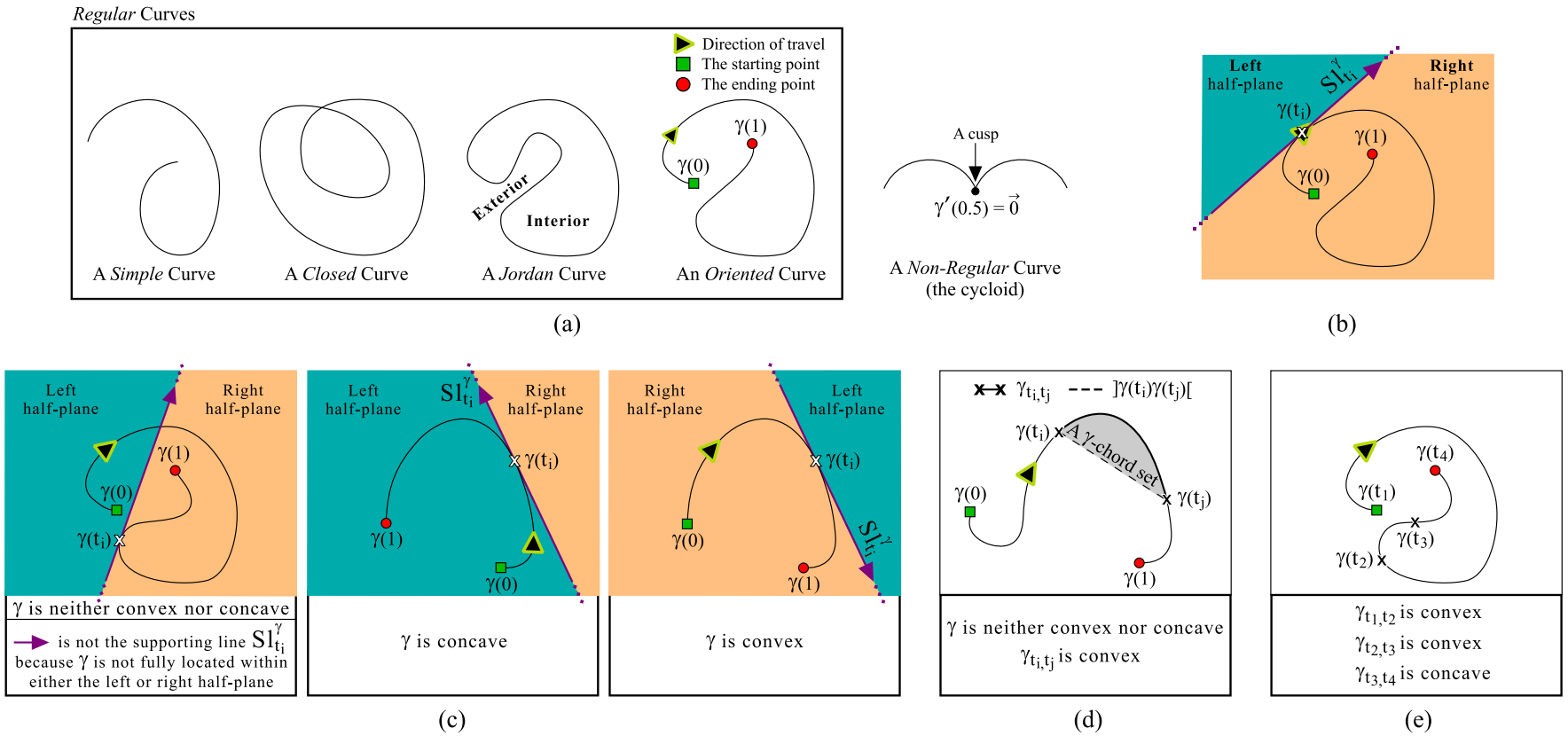


Figure 5.2: Several concepts about planar curves presented in visual form.

contains the entire curve in its left-labeled closed half-plane —consequently, condition 2 is not hold; and, to finish, the third curve is actually convex.

D4. Similarly to definition D3, an oriented planar curve γ is called *concave* if $\forall t \in \ ]0,1[$, (1) Sl_t^γ exists and (2) γ completely lies in the left-labeled closed half-plane of Sl_t^γ.

See the middle plot of figure 5.2(c) for an example of a concave curve.

D5. Let γ be an oriented planar curve, and let t_i and t_j designate any two values within the interval [0,1] such that $t_i < t_j$. Additionally, γ_{t_i,t_j} represents the oriented segment of γ starting at the point $\gamma(t_i)$ and ending at the point $\gamma(t_j)$. Finally, $]\gamma(t_i)\gamma(t_j)[$ means the open straight-line segment that joins the points $\gamma(t_i)$ and $\gamma(t_j)$.

With the above in mind, a *chord set* of γ is defined as the connected region enclosed by γ_{t_i,t_j} and $]\gamma(t_i)\gamma(t_j)[$, under the condition that γ_{t_i,t_j} is either convex or concave $\Big($look at figure 5.2(d)$\Big)$.

D6. Let γ be an oriented planar curve, and let γ_{t_i,t_j} have exactly the same meaning as in definition D5. Then, γ is a *piecewise convex-concave* curve if there exists a partition $0 = t_1 < \ldots < t_n = 1$ of the interval [0,1] such that: (1) $n \geq 2$; and (2) $\forall k\ 1 \leq k < n$, $\gamma_{t_k,t_{k+1}}$ is either convex or concave.

Observe that, for convenience, we do consider a convex curve and a concave curve as (degenerate) cases of a piecewise convex-concave curve —both of them given by $n = 2$, meaning thus that $\gamma = \gamma_{t_1,t_2}$.

By way of example, figure 5.2(e) depicts a piecewise convex-concave curve with $n = 4$.

5.1.2 Notation

In the following, we introduce the notation used throughout this chapter.

N1. $S, T \in \mathbb{R}^2$ are, respectively, the starting point and the target point of the robot's intended navigation task.

N2. $Cur \in \mathbb{R}^2$ specifies the current location of the robot on its way from S to T.

N3. Let XY and $]XY[$ —$X, Y \in \mathbb{R}^2$ and $X \neq Y$— define, respectively, the closed and open straight-line segment with endpoints X and Y (recall that a closed straight-line segment contains both endpoints, while an open straight-line segment contains none of them).

N4. $d(X, Y)$ is a function which measures the Euclidean distance between any two points X and Y —$X, Y \in \mathbb{R}^2$.

N5. As seen from equation 5.1, $sat(z)$ is a function which makes negative numbers to become zero —$z \in \mathbb{R}$.

$$sat(z) = \begin{cases} z & \text{if } \ z \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{5.1}$$

N6. $O_i \subset \mathbb{R}^2$ indicates a certain obstacle of the environment, and ∂O_i its planar contour curve that is supposed to be Jordan, regular, and piecewise convex-concave. With respect to ∂O_i, we further define:

- pL generically identifies a point that fulfills: (1) $pL \in \partial O_i$; and (2) the tangent line of the curve ∂O_i at the point pL passes through T.
- $I(\partial O_i)$ symbolizes the interior region of the Jordan-type curve ∂O_i.
- ${\partial O_i}^{+,X,Y}$ and ${\partial O_i}^{-,X,Y}$ designate, respectively, the positively- and negatively-oriented segment of ∂O_i joining the points X and Y —$X, Y \in \partial O_i$ and $X \neq Y$. By convention, a segment of ∂O_i with endpoints X and Y is considered to be positively —alternatively, negatively— oriented if $I(\partial O_i)$ is on the left —alternatively, right— hand side when traversing the segment from X to Y.

  Additionally, it is important to mention that, as derived from the initially-assumed geometrical properties of ∂O_i, the curve segments ${\partial O_i}^{+,X,Y}$ and ${\partial O_i}^{-,X,Y}$ are both ensured to be simple, open —that is to say, not closed—, regular, and piecewise convex-concave.

  In the remaining part of this section, ${\partial O_i}^{+,X,Y}$ and ${\partial O_i}^{-,X,Y}$ will be regarded as two oriented parametric curves. Therefore, as stated in definition D1, we will exploit the fact that such curves can be described by means of a continuous mapping ${\partial O_i}^{pm,X,Y} : [0,1] \rightarrow \mathbb{R}^2$ with pm being either $+$ or $-$. As one can easily guess, ${\partial O_i}^{pm,X,Y}(0) = X$, ${\partial O_i}^{pm,X,Y}(t)$ with $t \in \,]0,1[$, and ${\partial O_i}^{pm,X,Y}(1) = Y$ are, respectively, the starting point, the intermediate points, and the ending point of the curve.
- $Vv_{t_j}^{{\partial O_i}^{pm,X,Y}}$ with $t_j \in \,]0,1[$ and $pm \in \{+,-\}$ is a vector that has the point ${\partial O_i}^{pm,X,Y}(t_j)$ as its origin and points in the direction of travel of the curve ${\partial O_i}^{pm,X,Y}$ at ${\partial O_i}^{pm,X,Y}(t_j)$ —remember that ${\partial O_i}^{pm,X,Y}$ is a curve with a specific direction of travel because of being oriented; to be precise, this direction is the one that moves along the curve from ${\partial O_i}^{pm,X,Y}(0)$ to ${\partial O_i}^{pm,X,Y}(1)$.

  Alternatively, $Vv_{t_j}^{{\partial O_i}^{pm,X,Y}}$ can also be interpreted as the velocity / tangent vector to the curve ${\partial O_i}^{pm,X,Y}$ at the point ${\partial O_i}^{pm,X,Y}(t_j)$. More formally,

$$Vv_{t_j}^{{\partial O_i}^{pm,X,Y}} = {\partial O_i}^{pm,X,Y\,\prime}(t_j) = \lim_{h \to 0} \frac{{\partial O_i}^{pm,X,Y}(t_j + h) - {\partial O_i}^{pm,X,Y}(t_j)}{h}, \quad (5.2)$$

  where ${\partial O_i}^{pm,X,Y\,\prime}$ means the derivative of ${\partial O_i}^{pm,X,Y}$.
- Let us consider t_j, t_k, and pm such that $t_j, t_k \in [0,1]$, $t_j < t_k$, and $pm \in \{+,-\}$. Then, ${\partial O_i}_{t_j,t_k}^{pm,X,Y}$ corresponds to the oriented segment of ${\partial O_i}^{pm,X,Y}$ starting at the point ${\partial O_i}^{pm,X,Y}(t_j)$ and ending at the point ${\partial O_i}^{pm,X,Y}(t_k)$.
- Given t_j, t_k, and pm under the same restrictions as above, $Cs_{t_j,t_k}^{{\partial O_i}^{pm,X,Y}}$ denotes the chord set bounded by the curve segment ${\partial O_i}_{t_j,t_k}^{pm,X,Y}$ and the open straight-line segment $]{\partial O_i}^{pm,X,Y}(t_j){\partial O_i}^{pm,X,Y}(t_k)[$.

  Lastly, in this context, it is particularly relevant to note that one should be careful in using the term $Cs_{t_j,t_k}^{{\partial O_i}^{pm,X,Y}}$, since it can be just applied —as clearly imposed by definition D5— to cases where ${\partial O_i}_{t_j,t_k}^{pm,X,Y}$ is either convex or concave.
- As formally expressed by equation 5.3, φ is a function that returns either -1 or 1 depending on whether the interior region of ∂O_i contains the chord set $Cs_{t_j,t_k}^{{\partial O_i}^{pm,X,Y}}$ or not.

$$\varphi\Big(\partial O_i, Cs_{t_j,t_k}^{{\partial O_i}^{pm,X,Y}}\Big) = \begin{cases} -1 & \text{if } Cs_{t_j,t_k}^{{\partial O_i}^{pm,X,Y}} \subseteq I(\partial O_i) \\ +1 & \text{otherwise.} \end{cases} \quad (5.3)$$

To close, the previously introduced obstacle-related notation is graphically summarized in figure 5.3(a).

N7. Let u_X and v_Y be two vectors with origin in X and Y, respectively —$X, Y \in \mathbb{R}^2$. Then, the function $\theta(u_X, v_Y)$ consists of two main steps: (1) first of all, the vector u_X is translated so that its origin coincides with Y (as a result, we obtain a new vector u_Y); (2) after that, θ calculates and returns the smallest angle between u_Y and v_Y in absolute value $\Big($formally speaking, such a smallest angle is determined by the expression $arccos(\hat{u}_Y \cdot \hat{v}_Y)$, where $\hat{u}_Y$ and $\hat{v}_Y$ are, respectively, the unit vectors of u_Y and v_Y, and '$\cdot$' means the dot-product operation$\Big)$.

See figure 5.3(b) to better understand how the function θ works.

5.2 A Geometrical View of the Two Component Methods of $BugT^2$

A new non-pure reactive navigation strategy, named $BugT^2$, is put forward along this chapter by properly combining both the algorithm Bug2+ and the T^2 navigation framework. Next, these components of $BugT^2$ are geometrically described.

5.2.1 Regarding the Algorithm Bug2+

Section 4.2, and particularly algorithm 4.2, contains a complete geometrical description of Bug2+. Consequently, the reader is referred directly to this section for the details.

5.2.2 Regarding the T^2 Navigation Framework

In the following, we progressively explain and exemplify the geometrical model that underlies the T^2 navigation framework.

5.2.2.1 Description of the Model in terms of Specific Behaviors and Transitions

Like most *Bug*-type algorithms, our T^2 navigation framework does switch between two primitive behaviors, named *motion-to-goal* and *boundary-following*. In short, the motion-to-goal behavior is activated first and forces the robot to move straight towards the target (T). Moreover, this behavior remains active until finding an obstacle that impedes continuing the intended straight-line path to T. Under such a circumstance, in addition to stopping the motion-to-goal behavior, the boundary-following behavior is invoked so as to try to circumnavigate the blocking obstacle. In the context of T^2, the task of circumnavigation essentially consists in starting to follow the contour of the obstacle, and keep doing this contour following process until the two next conditions are both satisfied: (1) the robot is currently located at a pL-type point (refer to section 5.1.2 for information about this kind of strategic navigation points); (2) the so-called function Ω is zero when evaluated at the portion of the obstacle contour curve that has been traversed by the robot so far as a result of the ongoing contour following process. At the time that both of these conditions are met, the boundary-following behavior is replaced by the motion-to-goal behavior.

The motion-to-goal and boundary-following behaviors alternate the control of the robot in the way explained above until T is finally reached. Algorithm 5.1 provides a wider and

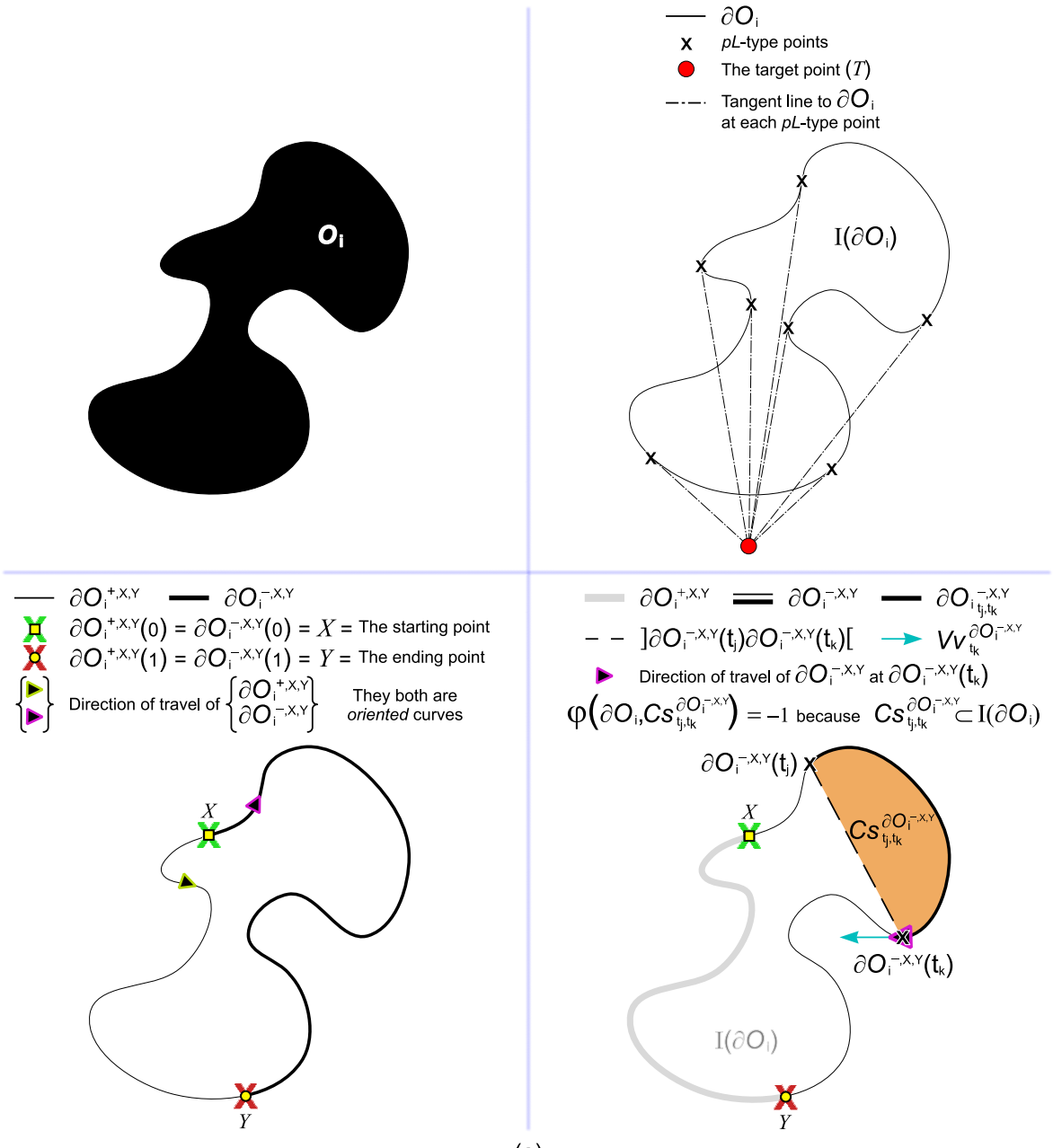


Figure 5.3: Illustration of part of the notation adopted for the geometrical description of the new algorithm $BugT^2$.

more formal description of the T^2 navigation framework. Notice that this description uses, but does not define, the aforesaid function Ω. To fill this void, the mathematical formulation of the function Ω is subsequently given and discussed.

5.2.2.2 A Deep Insight into the Ω Function

Parametrization

The Ω function is exactly as described in algorithm 5.2 (see at the end of the chapter). As can be observed, this function has four input parameters —in particular, these parameters are: ∂O_i, pm, X, and Y— which collectively specify some segment of the contour curve of the obstacle O_i. More to this point, the segment being specified is the one formally denoted as ${\partial O_i}^{pm,X,Y}$. As discussed earlier in section 5.1.2 —and graphically illustrated in figure 5.3—, ${\partial O_i}^{pm,X,Y}$ represents the pm-oriented segment of ∂O_i with endpoints X and Y. For the sake of clarity, notice that the expression 'pm-oriented' tries to emphasize the fact that the parameter pm allows indicating the orientation of the segment on which Ω is going to be performed (assuming that ∂O_i is a closed planar curve, there are two segments of ∂O_i having X and Y as endpoints; the parameter pm is used to unambiguously specify one of these segments). Just two values, $+$ and $-$, are actually considered for the parameter pm. Concisely, if $pm = +$ —meaning positive orientation—, ${\partial O_i}^{pm,X,Y}$ satisfies that, when moving along the segment from X to Y, $I(\partial O_i)$ is locally on the left-hand side; alternatively, if $pm = -$ —meaning negative orientation—, ${\partial O_i}^{pm,X,Y}$ satisfies the same as above with the only difference that $I(\partial O_i)$ is now lying on the right-hand side.

Operation

Roughly speaking, Ω is intended to calculate the total curvature —designated by tK in algorithm 5.2— of the segment ${\partial O_i}^{pm,X,Y}$ (as pointed out previously, this segment is uniquely specified through the set of function parameters). Or more explicitly, Ω does provide the sum of curvatures over all points of ${\partial O_i}^{pm,X,Y}$ (see footnote [6]).

Concerning the above, there are some important remarks to be made (these remarks reveal special aspects of the computation of the curvature-related sum carried out by Ω; additionally, notice that, in the text below, this sum is referred to as tK sum):

r1. The tK sum is a sum of signed curvatures. In this regard, figure 5.4 shows the two general cases that Ω uses as guidance for determining the sign of the curvature at each point of ${\partial O_i}^{pm,X,Y}$. As one could deduce from the figure, a point ${\partial O_i}^{pm,X,Y}(t)$ with $t \in \ ]0,1[$ is understood to have negative curvature if the chord set defined by $Cs_{t-\epsilon,t+\epsilon}^{{\partial O_i}^{pm,X,Y}}$ —being ϵ a positive infinitesimal value— is a subset of $I(\partial O_i)$; otherwise, the curvature of such a point is taken as positive.

r2. Let us consider the following situation: (1) the tK sum is being calculated, but it has not been yet completed $\Big($that is to say, Ω has summed the curvature values from a subset of the points of ${\partial O_i}^{pm,X,Y}\Big)$; besides, (2) the result of the partial tK sum in (1) is negative.

[6] The curvature of a parametric curve $\gamma : [0,1] \to \mathbb{R}^2$ at a point $\gamma(t)$ with $t \in \ [0,1]$ is to be the reciprocal of the radius of the circle that most closely approximates the curve near $\gamma(t)$

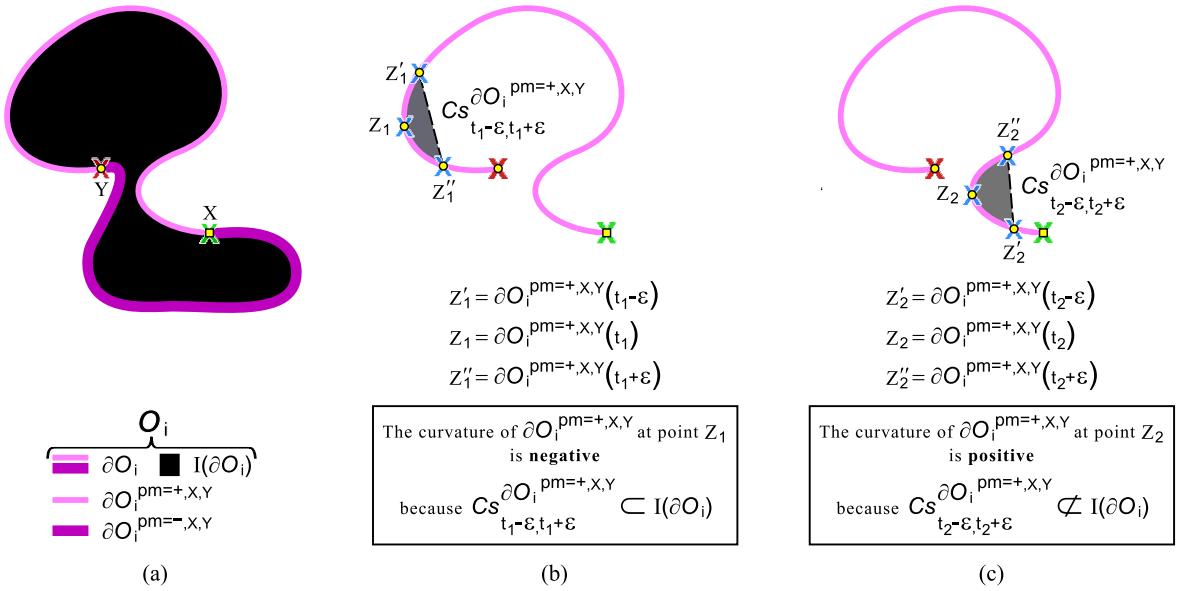


Figure 5.4: The sign of curvatures as interpreted by the Ω function: (a) the specific $\partial O_i^{pm,X,Y}$-type segment on which the two opposite-sign curvature cases are illustrated $\left(\text{to be precise, the segment chosen is the one given by } pm = + \text{ —or equivalently, } \partial O_i^{+,X,Y}\right)$; (b) case of negative curvature; (c) case of positive curvature. In (b) and (c), ϵ is considered an amount much larger than infinitesimal just for facilitating drawing.

During the computation of the tK sum, Ω is permanently checking whether the afore-considered situation occurs. What is more, when such a situation is known to happen, the currently computed partial tK sum is reset —i.e. set to zero.

Lastly, observe that, as a consequence of applying the preceding saturation process, we can trivially state that any partial tK sum is lower bounded by zero (as a side note, it should be mentioned that saturation is also applied to the result of the tK sum when it has been fully completed; therefore, the final tK sum is ensured to be 0-lower bounded as well).

r3. As one can easily deduce, remark *r2* makes relevant the particular order in which the curvature values are summed so as to get tK (depending on this order, the result of the tK sum may not be the same[7]). In this respect, it is worth clarifying that Ω performs the sum of curvatures following always the X-to-Y order of the points along $\partial O_i^{pm,X,Y}$. Or expressing this formally, we can equivalently say that: if $Z_1 = \partial O_i^{pm,X,Y}(t_1)$ and $Z_2 = \partial O_i^{pm,X,Y}(t_2)$ are two points of $\partial O_i^{pm,X,Y}$ such that $t_1, t_2 \in [0,1]$ and $t_1 < t_2$, then, in the tK sum, the term corresponding to the curvature value of Z_1 comes before the one of Z_2.

[7] Let us imagine a tK sum that involves adding, with saturation —in the way suggested by remark *r2*—, the three values as follows: 0.7, −0.9, and 0.1. Under these circumstances, one can readily obtain different results for the tK sum by varying the order in which the afore-specified values are summed. By way of example, $tK = 0.7 + -0.9 + 0.1 = \mathbf{0.1}$, and $tK = -0.9 + 0.7 + 0.1 = \mathbf{0.8}$

Implementation Issues

From an implementation viewpoint, Ω does not calculate the total curvature of ${\partial O_i}^{pm,X,Y}$ (tK) exactly in the manner discussed earlier, i.e. by summing / integrating the curvatures over all points of such a segment. Rigorously speaking, Ω computes tK by sequentially applying the next steps:

s0. As a preliminar step, the segment ${\partial O_i}^{pm,X,Y}$ is suitably partitioned into subsegments. Notice that, in the previous sentence, the word 'suitably' is specifically used to mean that the partition should be made in such a way that the curvatures of the points in each subsegment have all the same sign[8] (see remark *r1* for details about how to generally determine the sign of the curvature at a point of a plane curve).

s1. This first main step involves the calculation of a total curvature for each *s0*-provided subsegment of ${\partial O_i}^{pm,X,Y}$.

Let ${\partial O_i}_{t_l,t_q}^{pm,X,Y}$ —$t_l, t_q \in [0,1]$ and $t_l < t_q$— be one of the aforementioned subsegments of ${\partial O_i}^{pm,X,Y}$. Then, the sign and magnitude of the total curvature of this subsegment is given by:

- ◇ **Regarding the sign**. First of all, remember that the way of partitioning ${\partial O_i}^{pm,X,Y}$ guarantees that there are no changes in the sign of the curvature along ${\partial O_i}_{t_l,t_q}^{pm,X,Y}$. As one can guess, this non-changing curvature sign precisely corresponds to the sign of the total curvature of ${\partial O_i}_{t_l,t_q}^{pm,X,Y}$.
- ◇ **Regarding the magnitude**. It is obtained as the absolute angular difference between the velocity / tangent vectors at the endpoints of ${\partial O_i}_{t_l,t_q}^{pm,X,Y}$.

s2. This second / final main step simply consists of summing the subsegments' total curvatures calculated in step *s1*. As one can clearly see, the result of this sum does reflect the intended-to-compute total curvature of ${\partial O_i}^{pm,X,Y}$ (tK).

In addition to the above, there are some details to be highlighted with respect to the concrete manner in which the aforesaid tK-related sum is actually made. Before discussing these details, nevertheless, let us establish a general execution context for the current step *s2*. Fundamentally, let us suppose that, in step *s0*, ${\partial O_i}^{pm,X,Y}$ is divided into $q-1$ subsegments —$q \geq 2$— according to the q-point partition $0 = t_1 < \ldots < t_q = 1$ $\Big($as defined in section 5.1, ${\partial O_i}^{pm,X,Y}$ is considered to be a parametric curve that takes the form ${\partial O_i}^{pm,X,Y} : [0,1] \rightarrow \mathbb{R}^2$; given this, there is no doubt that any partition of the interval [0,1] suggests a way of dividing ${\partial O_i}^{pm,X,Y}$ into subsegments$\Big)$. Notice that, as a consequence of applying the earlier mentioned q-point partition on ${\partial O_i}^{pm,X,Y}$, the following subsegments are provided by step *s0*: $\forall l\ 1 \leq l < q$, ${\partial O_i}_{t_l,t_{l+1}}^{pm,X,Y}$. Going now to the next step, *s1* is known to be focussed on computing the total curvature of each of the $q-1$ ${\partial O_i}^{pm,X,Y}$'s subsegments resulting from step *s0*. In this regard, let tK^l — $1 \leq l < q$— symbolize the total curvature of the subsegment ${\partial O_i}_{t_l,t_{l+1}}^{pm,X,Y}$, as determined by step *s1*. With this notation in mind, it is patently obvious that step *s2* receives

[8] Or in other words, step *s0* decomposes ${\partial O_i}^{pm,X,Y}$ into convex and concave subsegments

as input, from step *s1*, the sequence of values $tK^1, \ldots, tK^{q-1}$. Finally, getting to the important point here, it should be noted that step *s2* does obtain tK by summing such tK^l values just as indicated below[9]:

- tK^l values are summed in order of increasing l. This hence means that $tK = tK^1 + \ldots + tK^{q-1}$.
- The sum of all tK^l values is made one by one following the order given above. Moreover, each time a new tK^l value is added to tK, the result of this addition is checked for being negative. If so, the currently accumulated sum of tK^l values is forced to be zero.

To conclude, figures 5.5, 5.6, and 5.7 present a complete algorithm 5.2-based example of how the Ω function works. The primary aim of this example is to make visually evident the implementation details that have been previously revealed about Ω.

5.2.2.3 Exemplifying the Correctness of the Geometrical Model

Figure 5.8(b) depicts the trajectory that resulted from executing a strategy, based on the T^2 navigation framework, on a real Pioneer robot moving in the scenario of figure 5.8(a). More specifically, in this experiment, the robot was autonomously driven through the environment in accordance with the strategy *Unvarying* T^2 (see section 3.3 and pay close attention to how this strategy chooses between the left and right contour following direction when finding obstacles in the robot's way). As can be observed in figure 5.8(b), the robot did have to avoid a G-shaped obstacle so as to successfully reach the desired target.

Figure 5.9 geometrically justifies, by means of five major steps derived from algorithm 5.1, the generation of the trajectory of figure 5.8(b).

5.3 The New Algorithm *Bug*T^2

Algorithm 5.3 summarizes the way we propose for advantageously merging the strategy Bug2+ and the T^2 navigation framework $\Big($from now on, with the aim of achieving a readable writing, we will refer to this framework as strategy T^2, or simply $T^2\Big)$. As can be observed, our proposal to merge such strategies is developed under the name of *Bug*T^2. Going into details, *Bug*T^2 exhibits the well-known motion-to-goal and boundary-following behaviors.

With respect to the motion-to-goal behavior, we merely have to say that it is exactly as in Bug2+ and T^2; therefore, this behavior is responsible of moving the robot straight from the current robot's position to the desired target, as long as no obstacle is found; notice additionally that, in the case of finding an obstacle that prevents the robot from progressing along such a straight-line path, the motion-to-goal behavior immediately deactivates, and gives the boundary-following behavior total control over the robot's motion.

As a first general idea regarding the boundary-following behavior, we should point out that this behavior constitutes the key component of the new algorithm *Bug*T^2, because being where the merging of the Bug2+ and T^2 strategies takes actually place. As its name indicates, the boundary-following behavior moves the robot around the currently detected obstacle

[9]The forthcoming two-key-point description on how Ω specifically performs the sum of tK^l values is inherently associated with remarks *r2* and *r3* of the previous section

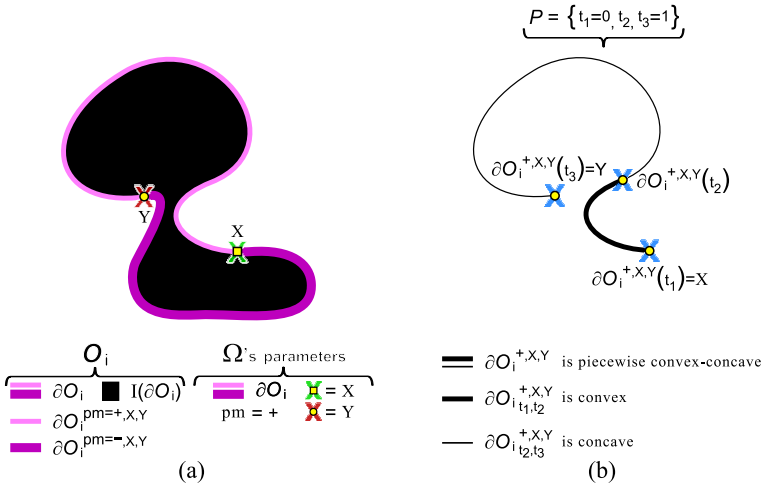


Figure 5.5: A working example of the Ω function as described in algorithm 5.2: (a) the concrete parameter setting being considered; (b) initializations (concisely, tK is initialized to zero —this is not really shown in the figure—, and the segment $\partial O_i{}^{+,X,Y}$ from (a) is minimally partitioned into convex and concave subsegments).

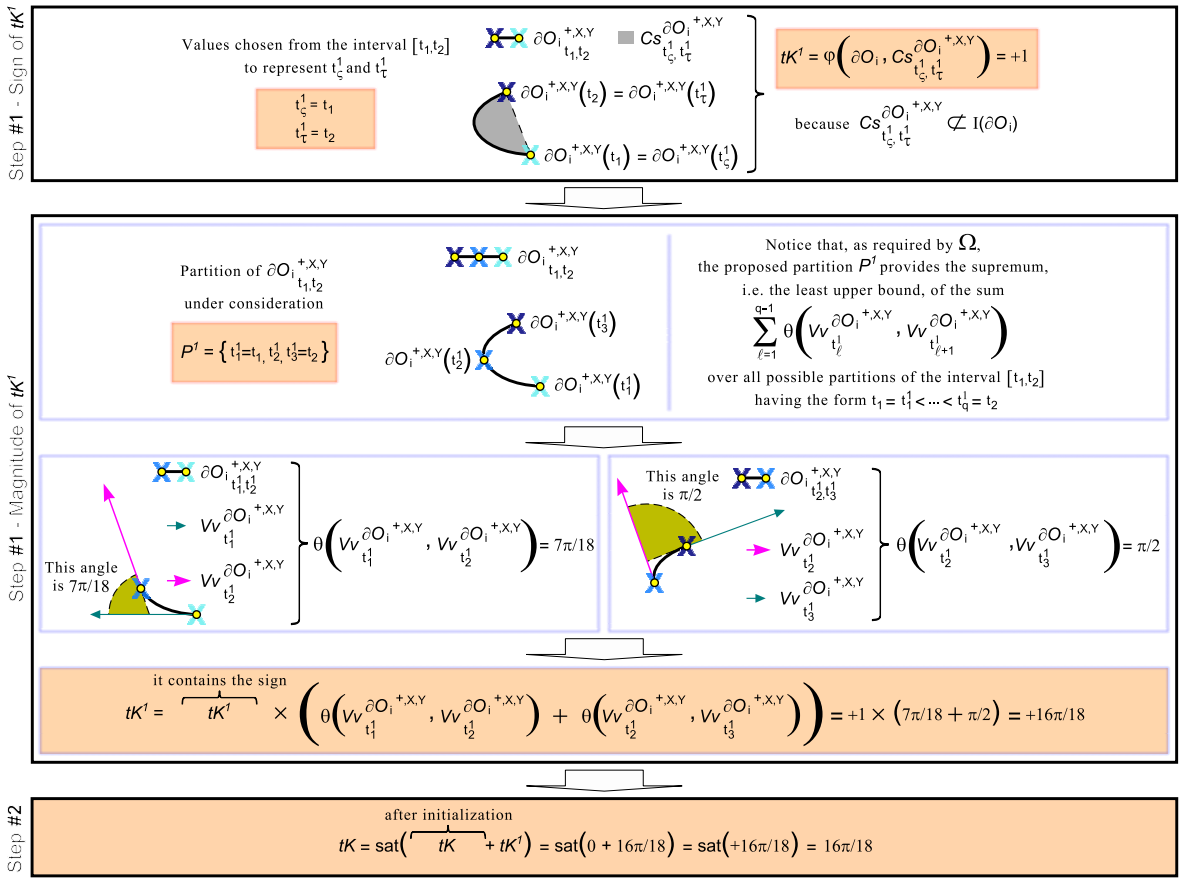


Figure 5.6: Continuation of the working example of Ω started in figure 5.5. The focus here is on illustrating, step by step, the operations performed in the first main loop iteration —$j = 1$— of the algorithm 5.2.

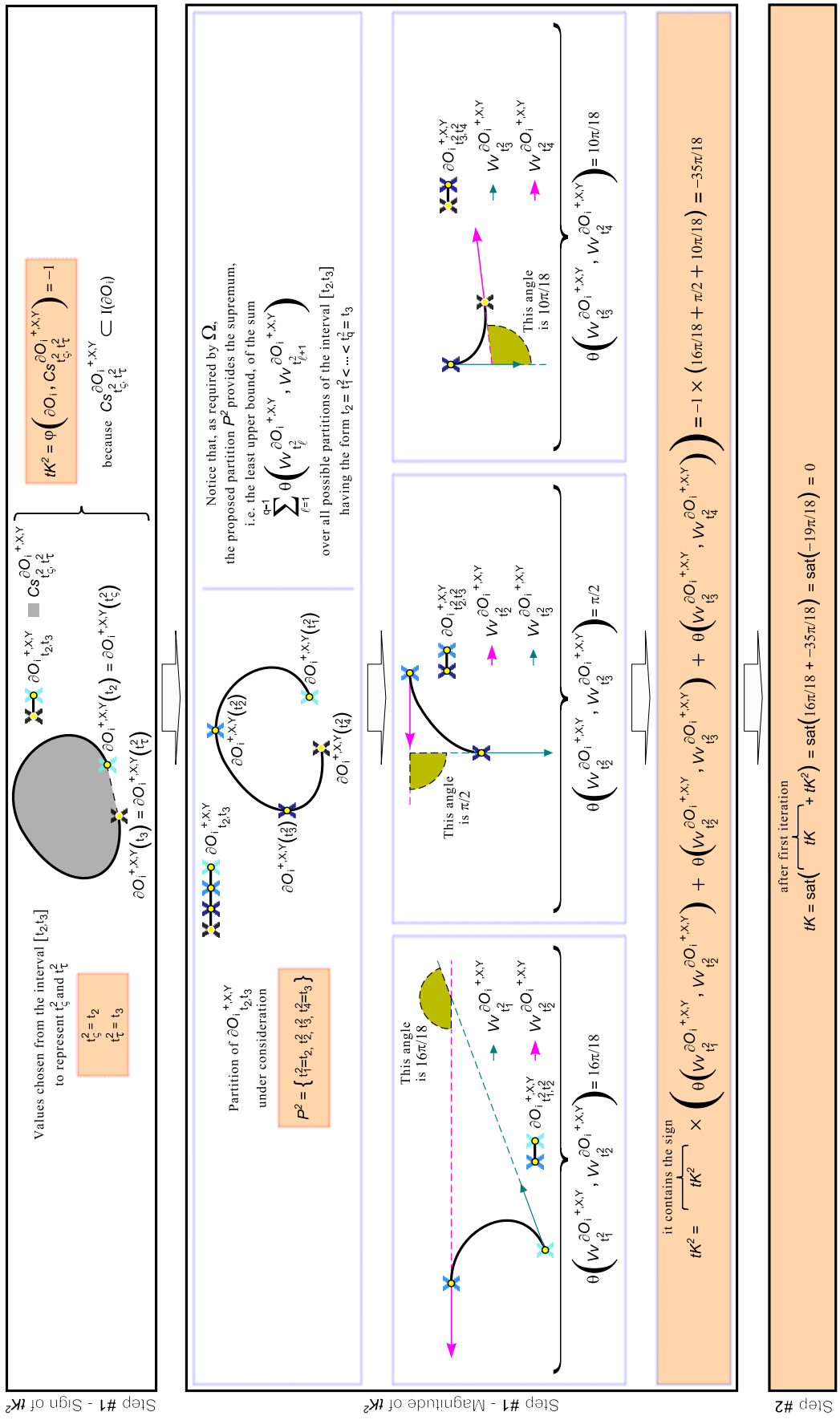


Figure 5.7: Continuation of the working example of Ω started in figures 5.5 and 5.6. The focus here is on illustrating, step by step, the operations performed in the second/last main loop iteration —$j = 2$— of the algorithm 5.2.

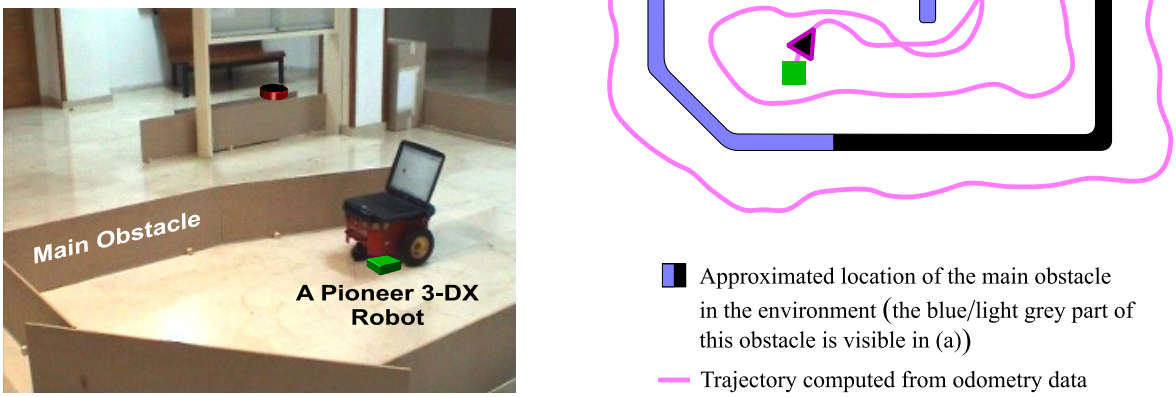


Figure 5.8: Navigation results of a real robot using the *Unvarying* T^2 strategy: (a) partial view of the environment set up with the robot placed at the starting position; (b) trajectory followed by the robot.

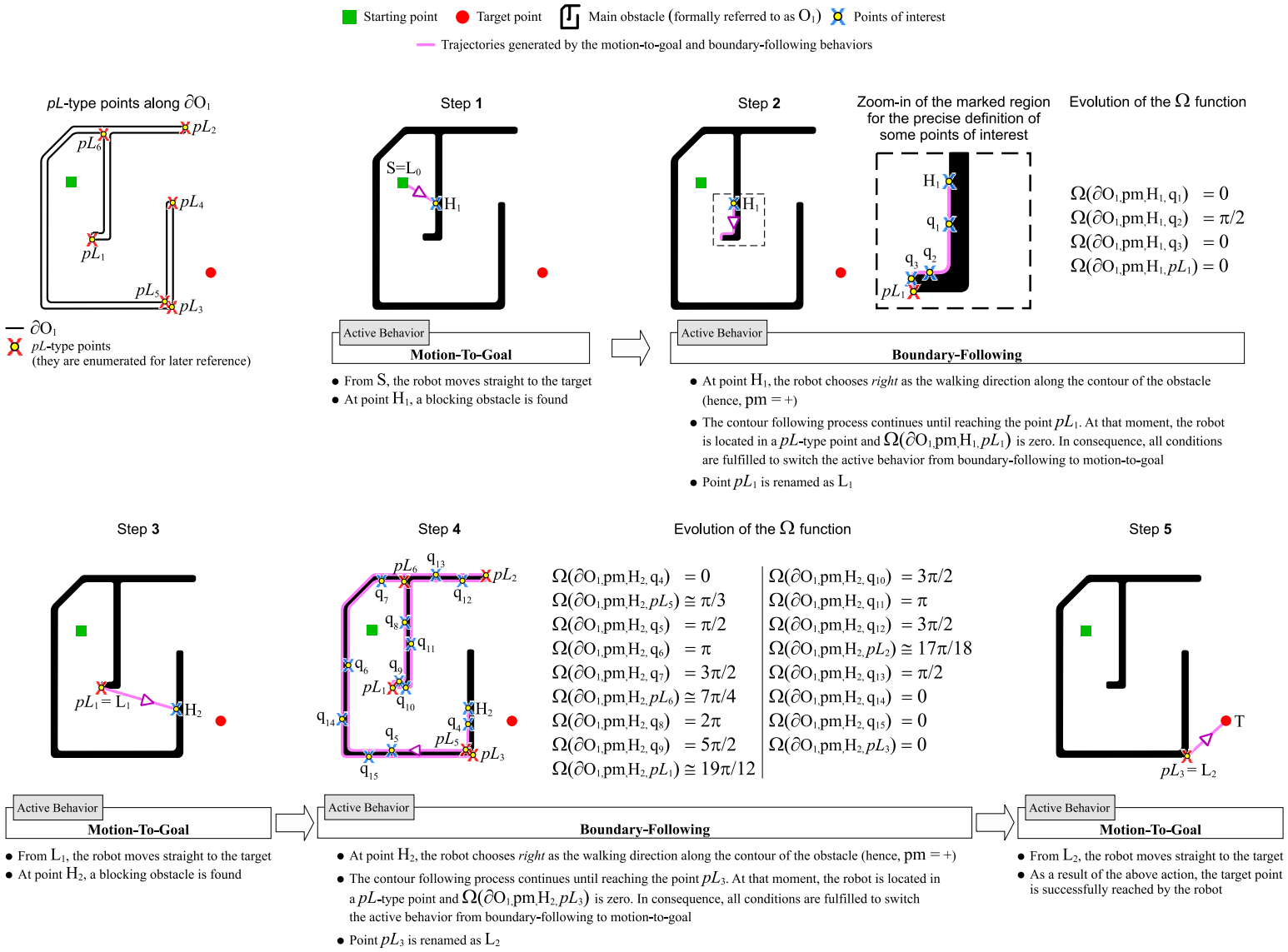


Figure 5.9: Application of algorithm 5.1 to geometrically analyze the T^2-generated trajectory of figure 5.8.

until a certain condition fulfills (in the related literature, this condition is formally known as *leaving* condition, with the purpose of stressing the fact that it determines the stop of the obstacle circumnavigation —and, consequently, it also involves reinvoking the motion-to-goal behavior). In $BugT^2$, the condition for leaving the boundary-following behavior is roughly divided into two parts: one part associated with achieving a reasonably good path length performance[10], and the other part focussing on guaranteeing global convergence (let us here make two pertinent remarks on these parts of the $BugT^2$'s leaving condition: (1) the performance-concerned part is based on the leaving condition that uses the strategy T^2, while the convergence-concerned part stems from the leaving condition of the strategy Bug2+; (2) a logical OR operator joins the performance- and convergence-concerned parts —this obviously means that the abandonment of the boundary-following behavior occurs when any, or both, of such parts is satisfied). Next, the two parts of the leaving condition of $BugT^2$ are briefly examined:

- ◇ **About the performance-concerned part**. In essence, this part does hold if, at the moment the robot reaches a non-previously-visited point of type pL, the Ω function happens to be zero (see section 5.1.2 to understand what a pL-type point signifies; on the other hand, notice that Ω is calculated on the portion of the obstacle contour that has been walked by the robot from the start of the boundary-following behavior until the achievement of the aforesaid first-time-visited pL-type point).

  The reader is referred to condition 2.c) of algorithm 5.3 for further information.

- ◇ **About the convergence-concerned part**. Omitting some minor details, this part does hold if the robot touches the open straight-line segment given by $]L_{j-1}T[$ —that is to say, the segment connecting the position where the boundary-following behavior was abandoned for the last time[11], and the target location.

  The reader is referred to condition 2.d) of algorithm 5.3 for further information.

To end, figure 5.10 shows the path followed by a $BugT^2$-based robot in a scenario where the target is surrounded by an intricately-shaped obstacle.

5.4 Analysis of the Standing out Properties of $BugT^2$

As outlined at the beginning of the chapter, the algorithm $BugT^2$ is proposed with the intention of making the most of both the strategy Bug2+ and the T^2 navigation framework. Or more precisely, $BugT^2$ pursues being a complete / convergent path planner — like Bug2+ —, as well as having a path length performance similar —in the majority of cases— to the one of the T^2 navigation framework. In the following, $BugT^2$ is demonstrated to possess the two previous valuable properties.

[10] As corresponds to a reactive navigation method, $BugT^2$ is designed to operate in unknown environments. In this context, notice that only a good / suboptimal path / solution can be expected to be obtained for the navigation problem at hand

[11] L_{j-1} is supposed to be the starting point —$L_{j-1} = S$— in the case that the boundary-following behavior has never been abandoned

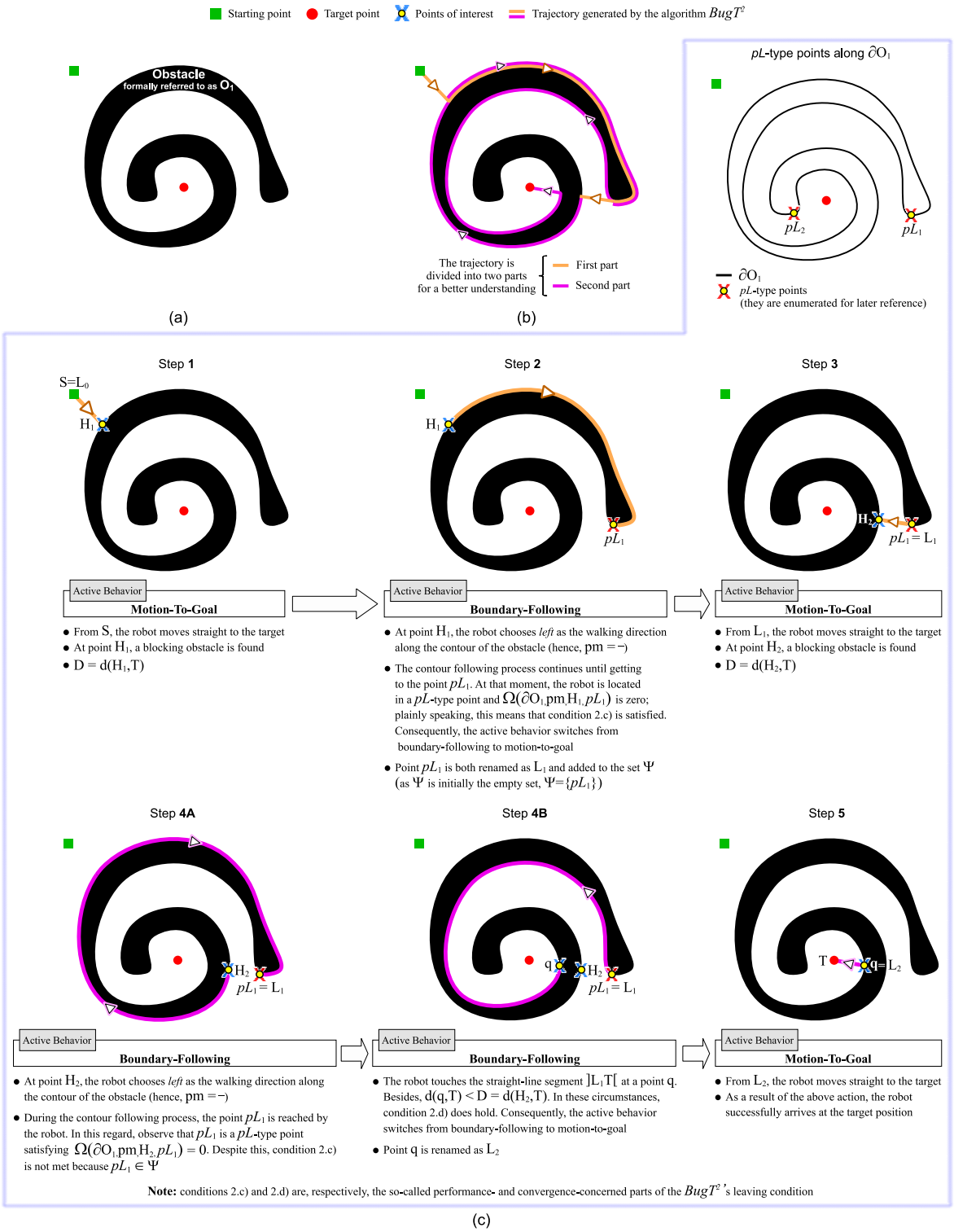


Figure 5.10: The algorithm $BugT^2$ solving a navigation task specially designed to try to trap the robot in an endless cyclic trajectory: (a) the environment set-up, composed by a loop-shaped obstacle; (b) resultant trajectory (notice that this trajectory does not come from a real test, but it has been obtained by theoretically applying algorithm 5.3, under the assumption that such an algorithm follows the obstacle boundaries to the robot's left); (c) the trajectory of (b) explained step by step.

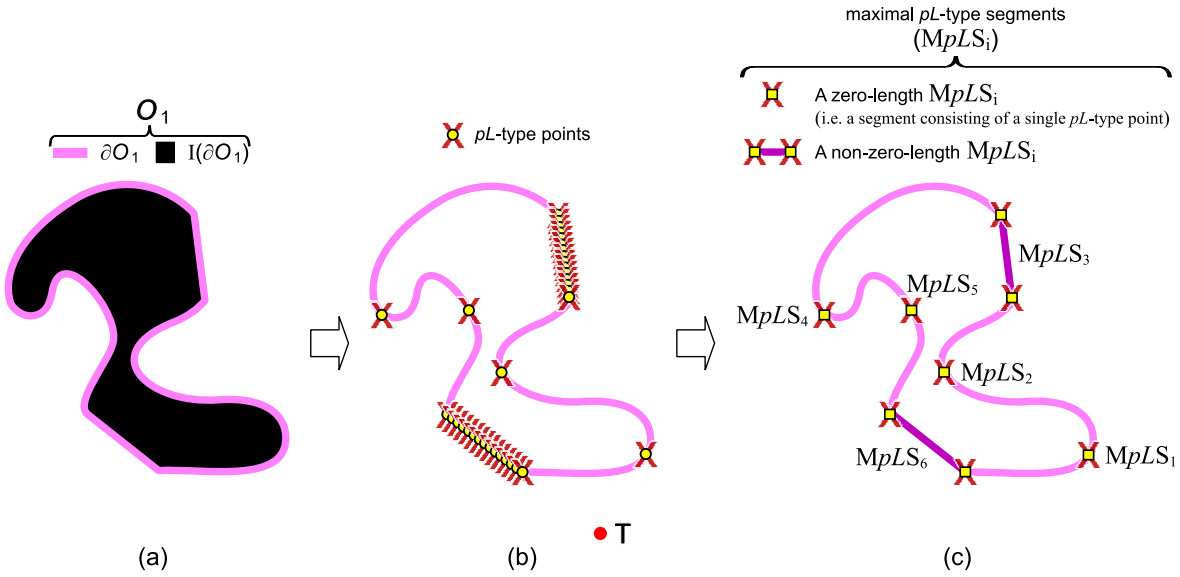


Figure 5.11: Concept of maximal pL-type segment: (a) obstacle being considered (O_1) and its contour curve (∂O_1); (b) pL-type points of ∂O_1 (observe that, to be able to determine these pL-type points, it has been necessary to fix a position for the target T); (c) maximal pL-type segments of ∂O_1.

5.4.1 Proof of Global Convergence

Under reasonable assumptions, lemmas 1, 2, and 3, and theorem 1 prove the convergence of the strategy $BugT^2$ —considering this strategy as described in algorithm 5.3.

Next, before proceeding with the statement and proof of the aforesaid lemmas and theorem, we will discuss the assumptions being made.

5.4.1.1 A Preliminary Definition

Let $\partial O_i : [0,1] \to \mathbb{R}^2$ be a parametric curve representation of the contour of a certain obstacle named O_i. Additionally, let us suppose that the point $\partial O_i(0) = \partial O_i(1)$ is not of type pL. At last, let $\partial O_{i\,t_j,t_k}$ denote the segment of ∂O_i that joins the points $\partial O_i(t_j)$ and $\partial O_i(t_k)$ —with $t_j, t_k \in\ ]0,1[$ and $t_j \leq t_k$.

Bearing the above in mind, it is said that $\partial O_{i\,t_j,t_k}$ is a *maximal pL-type* segment of ∂O_i if the following holds:

1. $\forall t\ t_j \leq t \leq t_k$, $\partial O_i(t)$ is a pL-type point;
2. $\inf\{t_j - t : 0 \leq t < t_j$ and $\partial O_i(t)$ is not a pL-type point$\} = 0$;
3. $\inf\{t - t_k : t_k < t \leq 1$ and $\partial O_i(t)$ is not a pL-type point$\} = 0$.

Finally, to illustrate the concept, figure 5.11 shows every maximal pL-type segment along the contour curve of a 2-shaped obstacle.

5.4.1.2 Assumptions About the Robot and the Navigation Environment

The foregoing proof of convergence of the strategy $BugT^2$ relies on the following assumptions:

- The ones made by the algorithm Bug2 / Bug2+ (see section 4.1.1). By way of summary, from all the assumptions of Bug2 / Bug2+, the most interesting to mention are:
  - ◇ T is reachable from S;
  - ◇ The environment is bounded, which means that the perimeter of any obstacle is finite, and the number of obstacles is finite as well;
- The contour of each obstacle is a Jordan, regular, and piecewise convex-concave planar curve that has only maximal pL-type segments of zero length[12].

5.4.1.3 Supporting Lemmas

Lemma 1. *The strategy $BugT^2$ converges to T if condition 2.c) is never satisfied*[13]*.*

Proof. If condition 2.c) is not met at any time, it can actually be removed from algorithm 5.3 without changing how this algorithm works at all. As an interesting point, notice that, after removal, the resultant algorithm appears to be identical to the one that was proposed in chapter 4 under the name of *Bug2+* $\Big($compare algorithm 5.3 —omitting condition 2.c)— and algorithm 4.2 to become aware that they both consist of exactly the same procedural steps$\Big)$.

Under the hypothesis of the non-fulfillment of condition 2.c), it seems clear from above that no difference exists between the strategies $BugT^2$ and Bug2+. Moreover, if, to this, we add the fact that the strategy Bug2+ is known to be convergent, there is no doubt that the strategy $BugT^2$ is convergent as well. □

Lemma 2. *The strategy $BugT^2$ converges to T if condition 2.c) is satisfied a number of times greater than zero and finite.*

Proof. Prior to giving the proof, we would like to make see that: first of all, let us suppose a situation where the control flow of algorithm 5.3 goes from step 2 to step 1 $\Big($notice that this step 2-to-step 1 transition reflects the moment in which algorithm 5.3 abandons the boundary-following behavior at a point that meets either condition 2.c) or condition 2.d)[14] — herein, let L_{j-1} be such a point$\Big)$; without loss of generality, the occurrence of the aforestated transition can be usefully understood as a restarting of algorithm 5.3 with the particularity of considering $S = L_{j-1}$ —or in other words, it is as if algorithm 5.3 were required to plan a path between L_{j-1} and T, irrespective of the previous path followed from the initial robot's location to L_{j-1}.

In the present lemma, the study of the convergence of the strategy $BugT^2$ is restricted to the case where condition 2.c) is fulfilled n times, being n a finite number greater than zero.

[12] Just to put an example, a curve with a continuously changing curvature does meet the requirement of exclusively having maximal pL-type segments of zero length

[13] As recently explained in section 5.3, condition 2.c) represents the so-called performance-concerned part of the leaving condition of $BugT^2$

[14] As recently explained in section 5.3, condition 2.d) represents the so-called convergence-concerned part of the leaving condition of $BugT^2$

To deal with this case, let us consider the moment in which condition 2.c) is satisfied for the last time. At such a moment, algorithm 5.3 should necessarily be executing step 2; what is more, as a consequence of meeting condition 2.c), algorithm 5.3 will proceed to step 1. In short, as it is obvious from the latter, the n^{th} occurrence of condition 2.c) will imply that algorithm 5.3 makes a transition from step 2 to step 1. In this respect, before drawing more definitive conclusions, it is important to recall the underlying meaning that, in algorithm 5.3, a step 2-to-step 1 transition really has. As we mentioned earlier, this transition is inherently equivalent to restart algorithm 5.3 in such a manner that the current robot's position is regarded as the point from where to begin looking for a path to the desired target (T).

Let $L_{j-1}^{2.c)^n}$ denote the position of the robot at the time that condition 2.c) of algorithm 5.3 holds for the n^{th}/last occasion. At $L_{j-1}^{2.c)^n}$, a transition from step 2 to step 1 is known to happen. Furthermore, by making use of the above-discussed way of interpreting these step 2-to-step 1 transitions, we can alternatively say that the reach of $L_{j-1}^{2.c)^n}$ causes that algorithm 5.3 does restart its execution under the assumption that $S = L_{j-1}^{2.c)^n}$ —and with no changes concerning the target point (T). Following further this line of thought, let us now examine some aspects of the new execution of algorithm 5.3 initiated after getting to $L_{j-1}^{2.c)^n}$. In this regard, notice that:

1. During this new execution of algorithm 5.3, it is ensured that condition 2.c) will never be satisfied, because the n^{th} / last occurrence of condition 2.c) has already taken place.

2. The planning of a free-obstacle path to the intended target location is certainly possible, because T is reachable from $S = L_{j-1}^{2.c)^n}$ $\Big($see footnote [15]$\Big)$.

As one can readily verify, points 1 and 2 are precisely the hypothesis under which the convergence of algorithm 5.3 was proved in lemma 1. Consequently, the same proof is valid for lemma 2. □

Lemma 3. *In the strategy $BugT^2$, condition 2.c) can only be satisfied a finite number of times.*

Proof. The two following facts can be directly inferred from analyzing algorithm 5.3:

f1. As an essential requirement to meet condition 2.c), the robot should be located at a pL-type point;

f2. Condition 2.c) cannot be fulfilled at exactly the same pL-type point more than once.

Alternatively, the assumptions made in section 5.4.1.2 do clearly imply that:

f3. The number of maximal pL-type segments per obstacle is finite, since the perimeter of each obstacle is supposed to be finite;

[15] First of all, with the aim of avoiding any confusion, let us consider that $S^\star$ represents the very initial position of the robot $\Big($such a position corresponds to the first value given to S; in relation to this, recall that there is a new value for S after each hypothetical restarting of algorithm 5.3 due to the fulfillment of condition 2.c) —or condition 2.d)$\Big)$. Then, in a scenario where T is reachable from $S^\star$ —as assumed under section 5.4.1.2—, and where the robot has succeeded moving from $S^\star$ to $L_{j-1}^{2.c)^n}$ —as being part of our line of reasoning—, it is undeniable that T can be attained from $L_{j-1}^{2.c)^n}$

f4. Taking into account both the previous fact (*f3*) as well as the widely- and here-adopted assumption that the number of potential obstacles is finite, there is no doubt that the total number of maximal pL-type segments —as summed over all obstacles— is also finite;

f5. As has just been revealed, the total number of maximal pL-type segments is finite. If, along with this, we consider the additionally-adopted assumption that any maximal pL-type segment is of zero length —and, therefore, consists of a single pL-type point—, it seems obvious that the total number of pL-type points is finite.

When putting together facts *f1*, *f2*, and *f5*, lemma 3 becomes trivial. □

5.4.1.4 Main Theorem

Theorem 1. *The strategy $BugT^2$ converges to T.*

Proof. On the one hand, lemmas 1 and 2 prove the convergence of the strategy $BugT^2$ for the cases where condition 2.c) is either never satisfied or is satisfied a finite number of times. On the other hand, from lemma 3, it is clear that condition 2.c) cannot hold for an infinite number of times; or using different words, such a lemma reveals that there are no remaining cases to consider —apart from the two mentioned above— in the proof of convergence of $BugT^2$. This proves theorem 1. □

5.4.2 *BugT*^{2} in Every Day Scenarios: Getting the Effective Path Length Performance of the T^2 Navigation Framework

Let us begin by defining what we understand as an *every day* scenario. Generally speaking, under the label of every day, we refer to a scenario that is formed by obstacles having non-natural/extremely-intricate shapes (notice that obstacles with unnatural/intricate shapes are, for the most part, artificially created to exhibit the properties of the navigation methods proposed). By way of example, figure 5.12 depicts a scenario of type every day, which corresponds to the ground floor of a grocery store located in New York. As can be seen, the subdivision of the store into areas gives rise to obstacles of relatively simple shape. On the other hand, figure 5.10 shows an example of a non-every day scenario $\Big($in this respect, it is important to stress that the convoluted shape of the only obstacle of figure 5.10 was purposely designed for illustrating the two situations that cause the leaving of the boundary-following behavior in the algorithm $BugT^2\Big)$.

When used in an every day scenario, the algorithm $BugT^2$ produces exactly the same path as would result from applying the T^2 navigation framework. This fact is well substantiated by the following four-step reasoning:

ed1. Scenarios that are not intricate, such as the so-called every day, allow the robot to choose among numerous alternative paths to reach the desired target (just to put an example, in the scenario of figure 5.12, there are far more than 16384 homotopically-different ways of achieving the target from the initial location).

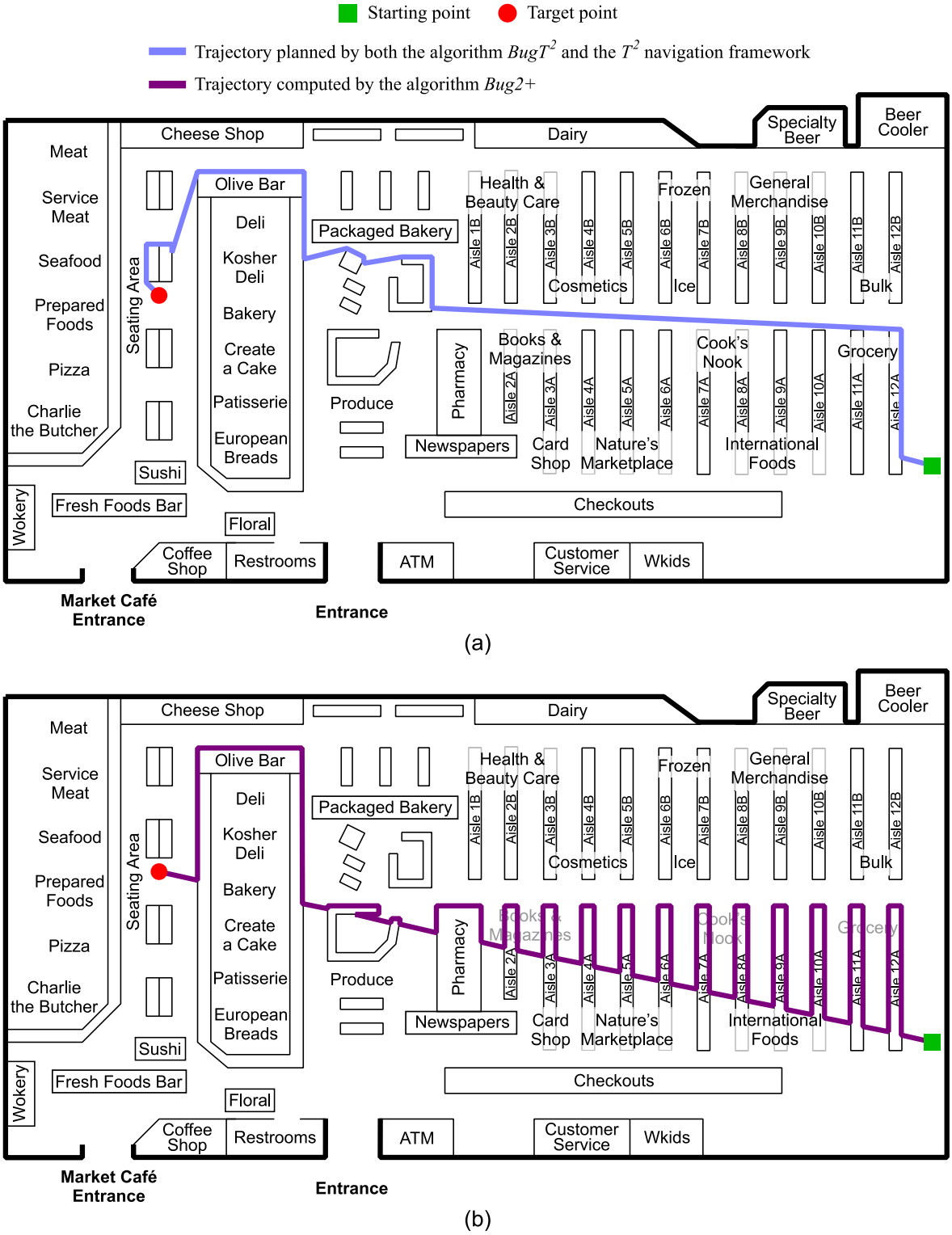


Figure 5.12: Evaluation of the path length performance of the approach $BugT^2$, the strategy Bug2+, and the T^2 navigation framework in a grocery-store environment: (a) path produced by both $BugT^2$ and T^2; (b) Bug2+ path. (The results of (a) and (b) are ideal paths, because they have been obtained by theoretically applying algorithms 5.3, 5.1, 4.2; notice additionally that, in all these algorithms, it has been assumed that the robot always follows the obstacle boundaries to its right).

ed2. The previous observation brings us to think that no matter which direction —in short, either left or right— the robot chooses for circumnavigating the obstacles that are found during the navigation, since such a choice will merely determine what of the huge amount of alternative paths to the target is finally generated.

Or in more conclusive words, we can state that, in the context of every day scenarios, decisions regarding the direction of obstacle circumnavigation do not negatively affect the successful attainment of the target position.

ed3. As widely discussed in section 5.3, the Bug2+-based component of the algorithm $BugT^2$ influences the navigation of the robot just when it is clear that the robot is not making progress towards the target. More specifically, our algorithm interprets that the robot is not progressing in meeting the target when coming back to an earlier-visited point —or to be precise, when returning to some pL-type point. Here, it is key to note that revisiting pL-type points is caused by errors committed when deciding about the proper direction of circumnavigation to take for successfully avoiding the obstacles detected by the robot. As an example of the latter, figure 5.10 illustrates how a robot, which navigates in accordance with the algorithm $BugT^2$, reaches the point pL_1 twice, essentially because of (wrongly) deciding to go around the obstacle O_1 in clockwise / left direction from point H_2; as can be seen in the figure, the direction of circumnavigation of O_1 at H_2 should have been counterclockwise / right to gain access to the target area.

ed4. Until now, our argumentation summarizes as follows:

- *ed1* and *ed2* explain that, in every day scenarios, there is not a preferable direction to circumnavigate an obstacle that is blocking the robot's intended path; or in other words, whichever the chosen direction —i.e. either left or right—, the robot will succeed in overcoming the obstacle.
- *ed3* establishes the circumstances in which the Bug2+-based component of $BugT^2$ becomes active. Concisely, the activation of this component requires that the robot has failed trying to avoid a certain blocking obstacle. In this respect, recall additionally that: (1) the algorithm $BugT^2$ recognizes that the afore-described failure has occurred when the robot visits an already-explored point of the obstacle; (2) the repeated visit of a point mentioned in (1) is due to having chosen a non-proper direction to go around the obstacle (as an important issue to keep in mind, notice that (2) suggests the existence of non-proper —and proper[16]— directions of circumnavigation for —at least, one of— the obstacles populating the scenario).

From above, it is evident that the switching on of the Bug2+-based component of $BugT^2$ requires obstacles whose avoidance may either fail or succeed depending on the direction taken for traveling around them. This necessity, however, directly contradicts the concept of every day scenario; as previously introduced, under this concept, any obstacle would have to be effectively overcome in any of the two possible directions of circumnavigation. In final conclusion, we can say that the Bug2+-based component of $BugT^2$ will never be switched on when navigating through an every day scenario. Consequently, in this

[16] Of the two possible directions to circumnavigate an obstacle, one of them is always proper under the condition that the target is reachable. In the present context, a *proper* direction of circumnavigation means that it allows avoiding the corresponding obstacle with success; or similarly, that it entails a step forward in achieving the target position

kind of scenarios, the navigation of the robot will be exclusively controlled by the other component of $BugT^2$, i.e. the one based on the T^2 navigation framework[17]. According to this, there is no doubt that, in scenarios labeled as every day, no difference can exist between the paths generated by the algorithm $BugT^2$ and the T^2 navigation framework (look at figure 5.12(a) for an example of this coincidence of paths).

In closing, we wish to emphasize that, when the scenario is of type every day —as usually happens in non-artificially created worlds—, the algorithm $BugT^2$ takes advantage of the non-excessive complexity of the obstacles to plan efficient-looking paths[18]. What is more, this is done by simply giving the T^2-based component of $BugT^2$ total control over the robot throughout the navigation[19] (comparing figures 5.12(a) and (b), one can clearly appreciate the big difference that, in terms of achieved path length performance, exists between permanently granting the responsibility of the robot's motion to one or other of the components of the algorithm $BugT^2$; the performance benefits of navigating solely in accordance with the T^2-based component are restricted to cases involving every day scenarios, that is to say, to cases where a T^2-like navigation does not put in risk, at any moment, the final convergence of the robot to the target position).

[17] In the remainder of this section, the term T^2-*based* will be used as a brief reference to the component of the algorithm $BugT^2$ that is rooted on the T^2 navigation framework

[18] As it is obvious, what path efficiency means depends on the context of the problem being addressed. In this chapter, we face the difficult task of autonomously navigating a robot to some final target point in an unknown terrain. In this particular context, we consider that a path is *efficient-looking* if it resembles the path that could be followed by an experienced blind person that is asked to move from the initial robot location to the target destination

[19] In $BugT^2$, the fact of putting the control of the robot in the 'hands' of the T^2-based component when navigating around an every day scenario is fully determined by the so-called leaving condition. The manner in which this condition is defined —see section 5.3— prioritizes performing T^2-like navigation against Bug2+-like navigation. Moreover, such a preference for the mode of navigation founded on T^2 is maintained until detecting that this mode no longer helps the robot in getting closer to the desired target (notice that this circumstance of lack of progress while navigating on the basis of T^2 never takes place in scenarios labeled as every day)

Algorithm 5.1 A description of the T^2 navigation framework in geometrical terms

Step 0:	Initializations
Step 1:	Motion-to-goal behavior
Step 2:	Boundary-following behavior

0) Initialize $j = 1$, and $Cur = L_0 = S$

1) Move along the straight-line segment $L_{j-1}T$ until one of the following occurs:

   a) T is reached. The algorithm stops

   b) An obstacle O_i is found. Define the hit point $H_j = Cur$, and go to step 2

2) Follow the contour of the obstacle (∂O_i) to either the *left* —i.e. in clockwise / negative direction— or *right* —i.e. in counterclockwise / positive direction— (see footnote $\star_1$). Set $pm = -$ if it is decided to choose *left* as the direction to go around ∂O_i from H_j; otherwise, set $pm = +$. Keep doing the contour following process until one of the next three possible situations arises:

   a) T is reached. The algorithm stops

   b) The robot returns to H_j. The algorithm stops because the target is unreachable (see footnote $\star_2$)

   c) The robot gets to a point —represented by Cur— that satisfies: firstly, Cur is of pL-type; secondly, $\Omega\Big(\partial O_i, pm, H_j, Cur\Big)$ is equal to zero; and, finally, the straight-line segment $CurT$ does not cross the obstacle O_i at point Cur. Under these circumstances, define the leave point $L_j = Cur$, increase j by one, and go to step 1

$\star_1$ From a generic perspective of the T^2 navigation framework, no criterion is explicitly put forward for selecting either the left or right contour following direction when finding a new blocking obstacle. With respect to this, nevertheless, it is important to emphasize that the way of making this selection is intrinsically associated with each specific T^2-based strategy, such as *Random* T^2 and *Unvarying* T^2. As widely discussed in chapter 3, the former strategy randomly chooses between left and right, while the latter strategy always chooses left or right

$\star_2$ Condition 2.b) is triggered when the robot completes a loop around the contour of the obstacle. Such an event certainly means that the target (T) is located inside the obstacle, being thus not possible to reach it.

With the above in mind, it is important to mention that condition 2.b) just allows the detection of some, and not all, of the situations where there is not a path from S to T

Algorithm 5.2 The function Ω described as a mathematical method

Parameters	∂O_i	∂O_i is the contour curve of a certain obstacle $O_i \subset R^2$
	pm	$pm \in \{+, -\}$
	X, Y	$X, Y \in \partial O_i$ and $X \neq Y$
Assumption		The function Ω assumes that ∂O_i is a Jordan, regular, and piecewise convex-concave curve
Consequences of the foregoing assumption		The segment of ∂O_i given by ${\partial O_i}^{pm,X,Y}$ is guaranteed to be a simple, open, regular, and piecewise convex-concave curve. Moreover, notice that, because ${\partial O_i}^{pm,X,Y}$ is a piecewise convex-concave curve, there exists a partition $P = \{t_1, \ldots, t_r\}$ of the interval [0,1] such that: (i) $0 = t_1 < \ldots < t_r = 1$; and (ii) $\forall j\ 1 \leq j < r$, ${\partial O_i}^{pm,X,Y}_{t_j,t_{j+1}}$ is either convex or concave

Initialize $tK = 0$

for $j = 1$ to $r - 1$ **do**

Step **#1**	Sign of tK^j	Calculate
		$$tK^j = \varphi\left(\partial O_i, Cs^{\partial {O_i}^{pm,X,Y}}_{t^j_\varsigma, t^j_\tau}\right),$$ where: • t^j_ς, t^j_τ are two values from the interval $[t_j, t_{j+1}]$ —$t_j, t_{j+1} \in P$— satisfying: (i) $t^j_\varsigma < t^j_\tau$; and (ii) $Cs^{\partial {O_i}^{pm,X,Y}}_{t^j_\varsigma, t^j_\tau}$ is entirely either inside or outside of $I(\partial O_i)$
	Magnitude of tK^j	Calculate
		$$tK^j = tK^j \ \times\ \sup_{P^j} \sum_{l=1}^{q-1} \theta\left(Vv^{\partial {O_i}^{pm,X,Y}}_{t^j_l}, Vv^{\partial {O_i}^{pm,X,Y}}_{t^j_{l+1}}\right),$$ where: • $P^j = \left\{t^j_1, \ldots, t^j_q\right\}$ denotes a partition of the interval $[t_j, t_{j+1}]$ —$t_j, t_{j+1} \in P$—, defined by $t_j = t^j_1 < \ldots < t^j_q = t_{j+1}$ • $\sup_{P^j}$ means the supremum over all possible partitions P^j The concept of supremum is a mathematical formalism that Ω employs to ensure that the following condition holds: $\forall l\ 1 \leq l < q$, the total curvature of ${\partial O_i}^{pm,X,Y}_{t^j_l, t^j_{l+1}}$, in absolute value, never exceeds π. Under this condition, such a total curvature can be well estimated by the expression $\theta\left(Vv^{\partial {O_i}^{pm,X,Y}}_{t^j_l}, Vv^{\partial {O_i}^{pm,X,Y}}_{t^j_{l+1}}\right)$

Step **#2**	Calculate
	$tK = sat\left(tK + tK^j\right)$

end for

return tK

Algorithm 5.3 $BugT^2$: A new method built upon the combination of the algorithm Bug2+ and the T^2 navigation framework

Step 0:	Initializations
Step 1:	Motion-to-goal behavior
Step 2:	Boundary-following behavior

0) Initialize $j = 1$, $Cur = L_0 = S$, and $\Psi = \emptyset$

1) Move along the straight-line segment $L_{j-1}T$ until one of the following occurs:

   a) T is reached. The algorithm stops

   b) An obstacle O_i is found. Define the hit point $H_j = Cur$, set $D = d(H_j, T)$, and go to step 2

2) Follow the contour of the obstacle (∂O_i) to either the *left* —i.e. in clockwise / negative direction— or *right* —i.e. in counterclockwise / positive direction. Set $pm = -$ if it is decided to choose *left* as the direction to go around ∂O_i from H_j; otherwise, set $pm = +$. Keep doing the contour following process until one of the next four possible situations arises:

   a) T is reached. The algorithm stops

   b) The robot returns to H_j. The algorithm stops because the target is unreachable

   c) The robot gets to a point —represented by Cur— that satisfies: firstly, Cur is of pL-type; secondly, $\Omega\Big(\partial O_i, pm, H_j, Cur\Big)$ is equal to zero; thirdly, the straight-line segment $CurT$ does not cross the obstacle O_i at point Cur; and, lastly, Cur is not in the set Ψ. Under these circumstances, add Cur to Ψ —$\Psi = \Psi \cup \{Cur\}$—, define the leave point $L_j = Cur$, increase j by one, and go to step 1

   d) The robot gets to a point —represented by Cur— that satisfies: firstly, Cur is a point of the open straight-line segment $]L_{j-1}T[$; and, secondly, $d(Cur, T) < D$. Under these circumstances, if the straight-line segment $CurT$ does not cross the obstacle O_i at point Cur, then define the leave point $L_j = Cur$, increase j by one, and go to step 1; otherwise, give to D the value of $d(Cur, T)$, and continue in step 2

The Use of Different *Bug*-like Strategies for Building Efficient Deterministic Anytime Path Planners

In the previous chapters —to be exact, from chapter 3 to 5—, three novel methods of reactive robot navigation have been formally put forward and successfully tested in both simulated and real experiments. This chapter goes one level beyond, focussing on two new algorithms of global path planning for environments that are completely known.

The chapter is structured as follows: first, to be clear about what was done in chapters 3–5 and what this chapter deals with, section 6.1 describes briefly the substancial differences between the problems of reactive navigation and global path planning[1]. Besides, the same section reveals the major shortcomings of global path planning, and states which of these shortcomings our algorithms do contribute to mitigate. Next, sections 6.2 and 6.3 explain in detail how our algorithms work.

6.1 Problem Definition and Objectives

6.1.1 Shift of the Focus From Reactive Navigation to Global Path Planning

A reactive navigation method is characterized by its inherent sense-and-act loop; more exactly, at each loop iteration, the method takes the local sensing data, plans the next action by applying simple procedures, and acts accordingly. In this way, these methods are able to respond rapidly to changes that take place in the world; as a main disadvantage, nevertheless, it has to be noted that, due to the local nature of the data in which they base their decisions, the successful completion of the navigation task is not necessarily guaranteed. (As an exception to this, it should be said that there is a class of reactive navigation methods, known as non-pure, where completeness may be actually gained, because of allowing that some global data —which is gathered while navigating— do influence the local decisions on acting. In this respect, however, it is appropriate to clarify that, despite the influence from global data pointed out before, non-pure reactive navigation methods continue fundamentally being local methods. As a last observation, notice that any local navigation method exhibits as an important disadvantage the generation of rather suboptimal paths).

In contrast to reactive navigation methods, a global path planner relies on an accurate global model of the robot's environment (this model can be either directly given or incrementally built during navigation). Based on this model, a global path planner is able to determine

[1] also known as *deliberative* navigation

an efficient free-obstacle path between the starting position and the intended destination position of the robot. As one can easily guess, the computation of such an efficient path requires a deep analysis of all of the available environment information. Consequently, this analysis used to be computationally intensive. The latter fact is particularly relevant, since it is the cause of global path planners being poorly responsive to environmental changes. When navigating through environments that dynamically change over time, the path that is initially planned by a global path planner may be invalidated by moving obstacles. If so, the global path planner should provide an alternative path for the robot. To this end, first the planner should update its world model to reflect the changes in the environment; then, a new path should be planned by using the already-updated world model. As it is evident from the description, these two steps are quite computationally demanding. Furthermore, because of this, it seems reasonable to think that the environment may usually change more rapidly than the capability of the planner to perform such two steps. In short, the (relatively) high rate at which changes occur in the environment, together with the (relatively) low rate at which the robot's path can be replanned / readapted, supports the idea that global path planners react late to environmental changes.

Summarizing the above-stated advantages and disadvantages of global path planners as compared to reactive navigation methods: a global path planner is good in producing an optimal / near-optimal path, but poor in reacting to moving / unexpected obstacles.

6.1.2 An Overview of What is being Proposed

6.1.2.1 Main Objective

The aim of this chapter is to present several algorithms for global path planning that are more reactive to unexpected / unplanned events than those previously reported in the literature.

The two here-presented algorithms, named *ABUG* and *vABUG*, allow defining the desired level of reactiveness. Moreover, depending on the reactiveness level that is chosen, our algorithms plan a path whose quality ranges from reasonably good to optimal / near-optimal. To be precise, reasonably good paths are obtained when *ABUG* / *vABUG* is configured to be highly reactive —i.e. when the time given for planning is very limited; alternatively, *ABUG* / *vABUG* gets better and better approximations of the optimal path as it is configured in a less reactive manner —i.e. as more time is available for planning.

6.1.2.2 More About Both the ABUG and vABUG Global Path Planners

Among the common features of the *ABUG* and *vABUG* algorithms, the following ones deserve to be highlighted:

1. *ABUG* and *vABUG* are *deterministic* planners. What is more, they both are further classified as *anytime*. As is well-known from the literature [73, 74, 75], a planner is said to be anytime when it possesses the ability of generating a series of alternative paths towards the robot's target destination; besides, as a restriction, these planners should guarantee that each newly-generated path in the series does represent an improvement, typically in the sense of being shorter in length than all paths that had been previously calculated.

   ABUG and *vABUG* are purposely designed to be run during a user-defined amount of time. Once this time is over, they provide as output the best / last path that has been found.

2. Just like any other deterministic planner, *ABUG* and *vABUG* are constrained to be applied to low-dimensional path planning problems. More precisely, they can be used for finding paths in two- and three-dimensional configuration spaces. Notice, however, that, under two and three dimensions, we can express the majority of problems of path planning in the growing area of low-cost robotics applications, e.g. for the home appliances, entertainment, and educational sectors.

3. *ABUG* and *vABUG* explore the configuration space by making moves that are inspired by the biological behavior of some bugs / insects. As is well-known, many insects employ the direction of the sun's rays as a sort of compass. Based on this compass, these insects are able to maintain a correct heading; moreover, they are surprisingly able to travel from place to place and steer around obstacles.

   ABUG and *vABUG* plan paths by exploiting the potential that the above insects have for navigating between fixed points while avoiding obstacles. Concisely, *ABUG* and *vABUG* perform the task of path planning as follows: to start with, a colony of insects is virtually created; later, the insects forming part of the new colony are placed in the search space with the assigned task of moving from the initial robot configuration to the target robot configuration (in this way, for each insect, an alternative[2] path between such configurations is expected to be obtained).

   Finally, it should be mentioned that different types of insects are considered by *ABUG* and *vABUG*. In *ABUG*, insects are supposed to perceive the world around them through exclusively tactile sense organs. In *vABUG*, nevertheless, insects rely on visual sensing. As one can imagine, the use of different senses —tactile versus visual— means to have different views of the world; what is more, different views of the world necessarily result in different behaviors. In this regard, it is important to note that the tactile-based insects of *ABUG* behave according to the algorithm *Bug2+* (see chapter 4), while the vision-based insects of *vABUG* behave as described by the algorithm *VisBug* (refer to section 2.6.3).

6.2 *ABUG*: A Fast Anytime Path Planner Inspired in the Biological Behavior of Insects with Tactile Sensing

This section is organized as follows: section 6.2.1 briefly reviews the *Bug2+* strategy presented in chapter 4, while section 6.2.2 shows how Bug2+ can be used to construct the fast anytime global path planner named *ABUG* (besides, this section formally analyzes the properties of *ABUG*); section 6.2.3 reports some experimental results of our algorithm, and compares these results with those obtained by some techniques that are well-known in the field of path planning; and, lastly, section 6.2.4 draws some conclusions.

6.2.1 The Algorithm *Bug2+*

The anytime approach *ABUG* plans paths based on *Bug2+*, an enhanced version —suggested in this dissertation— of an algorithm called *Bug2* which was put forward by Lumelsky and Stepanov in [53]. This new *Bug*-derivative strategy preserves the simplicity as well

[2] Insects of the same type may not behave exactly the same under the same circumstances. This fact is captured by *ABUG* and *vABUG*, where each insect provides a different solution to the path planning problem

as the intuitive behavioral description by which the algorithm Bug2 is mostly characterized. Furthermore, the length of a path produced by Bug2+ is always less or equal to the one of Bug2 (in terms of path length performance, several scenarios have been constructed where Bug2+ has behaved over 26% better than Bug2). The strategy Bug2+ is a sensor-based path planner with proven termination conditions for environments which are static, unknown, and two-dimensional. In the following, the proposal is summarily discussed underlining its major points of discrepancy with regard to the classic version of the algorithm Bug2. See section 4.2 for a deeper explanation of Bug2+ as well as the formal proofs for the above-mentioned features of the strategy.

6.2.1.1 Notation

$S, T \in \mathbb{R}^2$ are, respectively, the starting and the target points of the mission. XY ($X, Y \in \mathbb{R}^2$ and $X \neq Y$) represents the straight-line segment with end points X and Y. The line connecting the starting and the target points, ST, is referred to as *main line*, or *m-line* in brief. On the other hand, O_i denotes a certain obstacle of the environment and ∂O_i its contour curve. Finally, $d(X, Y)$ is a function which measures the Euclidean distance between any two points X and Y ($X, Y \in \mathbb{R}^2$).

6.2.1.2 Description

The algorithm Bug2+ exhibits two different behaviors: motion-to-goal and boundary-following. During the former, which is activated first, the robot moves towards the target (T) along the m-line[3]. The boundary-following behavior, on the other hand, is invoked when the robot encounters an obstacle (O_i) on its way. The point where this obstacle is found is named hit point (H_j). Next, the robot follows the contour of the obstacle (∂O_i) to the left or right according to a user-definable parameter called *pCFD*. During this contour following process, a special situation may occur, in which the robot returns to H_j meaning that a loop around the obstacle boundary has been completed. In such a case, the target is inside the obstacle, not being thus achievable. More usual is, however, the situation where the robot reaches a new point on the m-line closer to T than H_j. At that moment, a leave point (L_j) is defined and the motion-to-goal behavior is invoked again.

Algorithm 4.2 provides a formal description of Bug2+, highlighting, in **bold**, the most important changes which have been done regarding the strategy Bug2. In short, the strategies Bug2 and Bug2+ use a different criterion to invoke the motion-to-goal behavior when the robot is circumnavigating the contour of an obstacle. Specifically, in Bug2 (see algorithm 4.1), this transition occurs when a point Q on the m-line nearer the target than H_j is found. Moreover, for really abandoning the boundary-following behavior, the point Q should satisfy an additional condition, which requires the robot to be able to move along the straight-line segment QT without immediately hitting the current obstacle —hereafter, let this condition be referred to as $C_{progress}$[4]. The strategy Bug2+ differs from Bug2 in considering the points which do not meet condition $C_{progress}$ into the decision associated with leaving the contour following process. Let Γ denote the set of m-line's points not fulfilling condition $C_{progress}$ which have been found by the robot during the last —and still current— activation of the boundary-following behavior. Then, the strategy Bug2+ will perform a transition to the motion-to-goal behavior when reaching a point Q on ST with $Q \notin \Gamma$ and satisfying the inequality $d(Q, T) <$

[3]The mobile robot is supposed to be a point fitted with a complete set of error-free tactile sensors

[4]Condition $C_{progress}$ appears under the name of C_2 and C_4 in algorithms 4.1 and 4.2, respectively

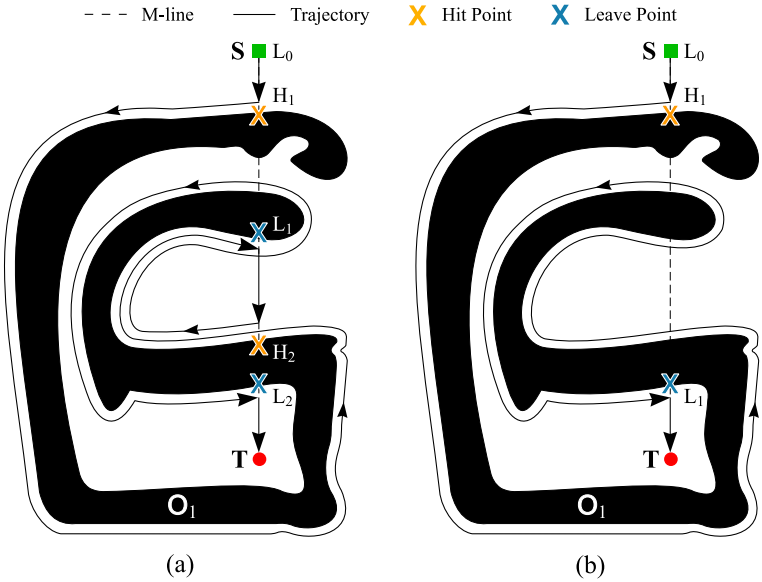


Figure 6.1: Comparison of the paths generated by (a) Bug2 and (b) Bug2+ in a scenario with an intricate obstacle. In both cases, the parameter *pCFD* was assumed to be *right*.

$min\{d(\gamma, T)\ \forall\gamma \in \Gamma\}$ (notice that $H_j \in \Gamma$ according to the definition of hit point). Figure 6.1 illustrates with an example the differences between the strategies Bug2 and Bug2+.

The length of a path planned by Bug2+ never exceeds the limit given by expression 6.1, where i denotes an obstacle of the scene (O_i), n_i represents the number of intersections of ∂O_i with the m-line, and B_i refers to the O_i's perimeter.

$$d(S,T) + \sum_i \frac{n_i}{2} B_i \tag{6.1}$$

6.2.2 The Algorithm *ABUG*

The algorithm Bug2+, as was described in section 6.2.1, allows us to plan a single path in an unknown and static environment. *ABUG* uses Bug2+ to generate multiple paths in an a priori known scenario just by considering both alternatives, left and right, whenever an obstacle is found, instead of keeping the parameter *pCFD* constant. Such a flexibility in the strategy Bug2+ does not jeopardize its convergence nor the rest of its properties. Figure 6.2 illustrates the above-mentioned exhaustive search in a simple scenario where four topologically different paths are planned.

Next, a deeper description of *ABUG* embedded into an A* framework [76] is presented. Additionally, a fast mode of operation based on inflating the heuristic cost function of the A* search is also put forward. Finally, the theoretical properties of the proposal are set out.

6.2.2.1 Description of the Planner

As can be observed at the bottom of figure 6.2(a), the algorithm *ABUG* makes use of a binary tree to search for paths, where each node represents a point of the environment in which a decision must be made regarding the direction —left or right— to be taken during the subsequent contour following process. On the other hand, taking into account both that

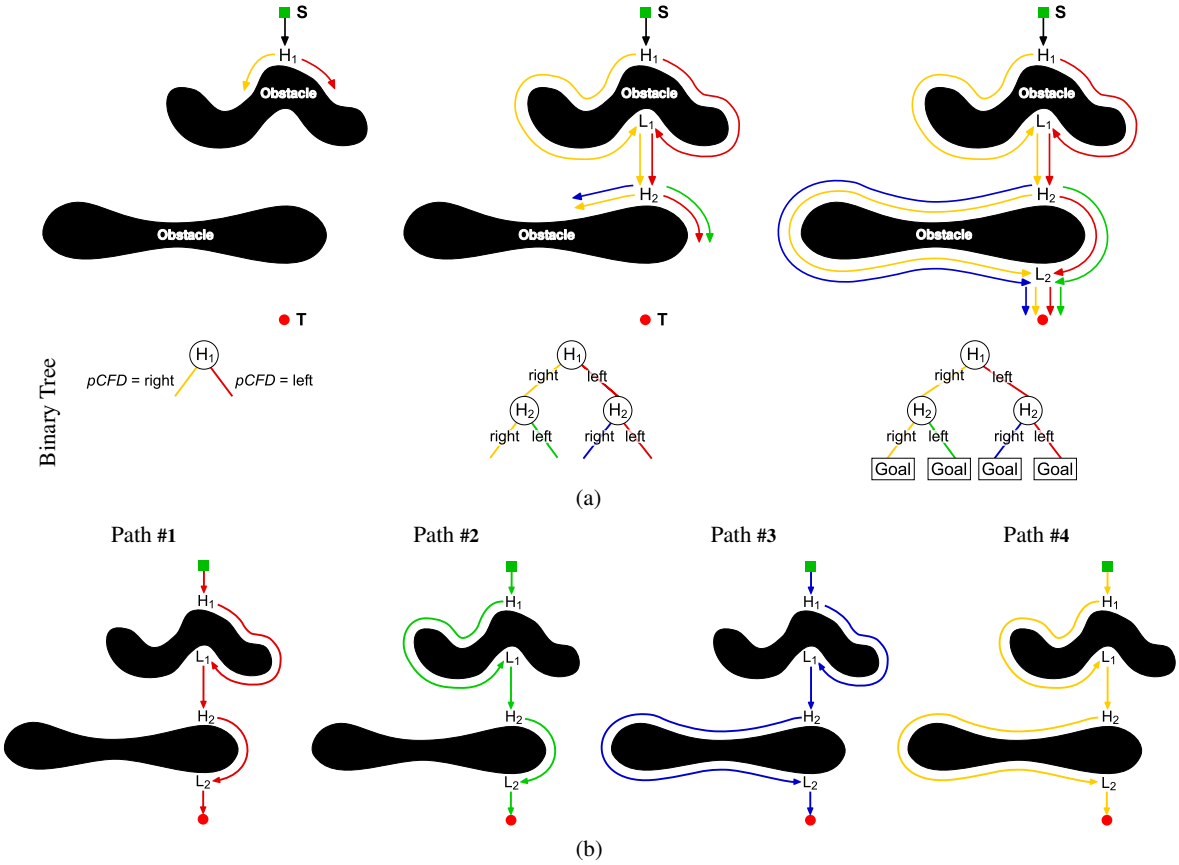


Figure 6.2: Exemplifying how the algorithm Bug2+ can be used for planning multiple paths: (a) the search step by step; (b) planned paths.

a binary tree is a particular case of a graph and that A* is a well-known and efficient method for exploring graphs, *ABUG* adopts A* for conducting the search of all Bug2+-compliant solutions. To this end, we define an estimated cost function f that returns as output the sum of two values: g and h. Specifically, given a still incomplete Bug2+ path P, g denotes the current P length and h the expected additional distance to be traveled until achieving the target (T). This distance is assumed to be the Euclidean, which means that *ABUG* applies an optimistic / admissible —and also consistent [77]— heuristic h, ensuring thus the optimality of the planner (or in other words, the shortest path within the graph will be found). Nodes / Paths are expanded / extended in the order of increasing f-values by means of a priority queue. The search starts with a degenerated path merely containing the starting point (S). This path is later prolonged on the basis of step 1 in algorithm 4.2, which involves moving straight towards T until finding an obstacle. At that moment, the resultant path is duplicated and, next, both are extended by following the boundary of the obstacle in opposite directions. Finally, once for a path the contour following process has finished (step 2 in algorithm 4.2), the path is placed into the aforementioned priority queue —*qPrio*— in a position in accordance to its updated f-value. The strategy continues by taking out from *qPrio* the estimated least-cost solution —the one with the minimum f-value— as well as by applying on it the preceding actions. Algorithm 6.1 describes formally the approach.

Algorithm 6.1 *ABUG*: A description in pseudocode

$\boldsymbol{f}(P)$
1: **return** $g(P) + h(P)$

Store(P)
2: Declare / Initialize the *static* variable $P_{Shortest}$ to NULL
3: **if** $P_{Shortest}$ == NULL **or** $g(P) < g(P_{Shortest})$ **then**
4: Set $P_{Shortest} = P$
5: **output** $\leftarrow P$
6: {a new solution P has been found; the planner continues looking for shorter paths}
7: **end if**

Improve(P)
8: $P_{Improved} = ShortestHomotopicPath(P)$
9: $Store(P_{Improved})$

Extend(P, $qPrio$)
10: *Motion-To-Goal*(P)
11: **if** T has been reached **then**
12: $Store(P)$; $Improve(P)$
13: **else** {An obstacle has been found}
14: **for** $pCFD$ = *left* **to** *right* **do**
15: Copy P into P_{New}
16: *Boundary-Following*(P_{New}, $pCFD$)
17: **if** T has been reached **then**
18: $Store(P_{New})$; $Improve(P_{New})$
19: **else if** T is unreachable **then**
20: Stop search
21: **else** {Conditions C_3 and C_4 of algorithm 4.2 have been met at Q}
22: Insert P_{New} into $qPrio$
23: **end if**
24: **end for**
25: **end if**

Main()
26: Build a path P consisting of only the starting point (S)
27: Insert P into $qPrio$
28: **repeat**
29: Pick P_{Best} from $qPrio$ such that $f(P_{Best}) \leq f(P)$, $\forall P \in qPrio$
30: Remove P_{Best} from $qPrio$
31: $Extend(P_{Best}, qPrio)$
32: **until** $qPrio$ is empty **or** T is known to be unreachable

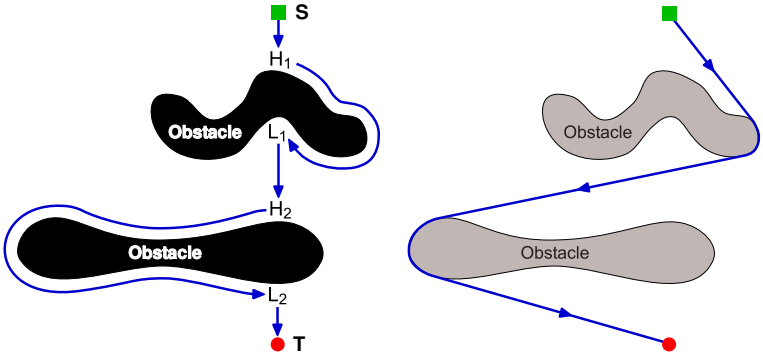


Figure 6.3: The shortest homotopic path problem: (a) a path; (b) the shortest path preserving the homotopy class of a).

It is important to note that *ABUG* is an anytime approach and not a classic planner that generates the best / shortest Bug2+ path as could be guessed so far. Anytime path planning, as pointed out in section 6.1, requires the progressive improvement of the quality of the solutions while time permits and the optimal path is not found. With this purpose, the algorithm *ABUG* makes use of the mathematical / topological concept of *path homotopy*, which provides us with an efficient way for optimizing a given path by solving the so-called *shortest homotopic path* problem [72]. Informally speaking, a path is regarded as an elastic band joining points S and T which is tightened to shorten it (see figure 6.3 for an example). Many methods have been proposed to compute the shortest homotopic path in $\mathbb{R}^2$. Among all of them, the one published in [78] has been finally applied because it presents the minimum —to the best of our knowledge— algorithmic complexity ($O(log^2\ n')$ per output vertex being n' the number of obstacles in the environment).

The search performed by *ABUG* does not stop after finding and, later, improving the best Bug2+ path. The strategy actually considers all the solutions within the graph induced by the binary tree data structure which is built. As will be seen in section 6.2.2.3, the number of solutions to be considered is bounded by a value that increases according to the complexity of the mission, although it grows in a reasonable way. The main reason for exploring the whole space of solutions is due to the fact that worse Bug2+ paths can become better solutions after being optimized as it so happens in the scenario of figure 6.4. Finally, observe that, each time a path is either computed or optimized by the algorithm *ABUG*, the new solution is compared with the best / shortest path found so far, and only if the former improves the latter, the strategy provides as output the new solution —i.e. line 5 in algorithm 6.1 is executed. Otherwise, the path is simply discarded. By applying such a filtering process, *ABUG* guarantees the generation of a strictly monotonically decreasing sequence of solutions regarding path length.

By way of notation, from now on, each of the Bug2+-compliant paths found by the strategy *ABUG* before being optimized will be denoted as π_k, where the index k is a sequence number that indicates the order in which the path was obtained. On the other hand, let $\Pi = \{\pi_1, \ldots, \pi_q\}$[5] be a set containing all these solutions. Notice that, as a direct consequence of the heuristic defined by the planner, π_k is shorten or equal than π_l if $k < l$.

[5] Π is the empty set when there is not a solution to the path-planning problem; that is to say, when the target point (T) is not reachable

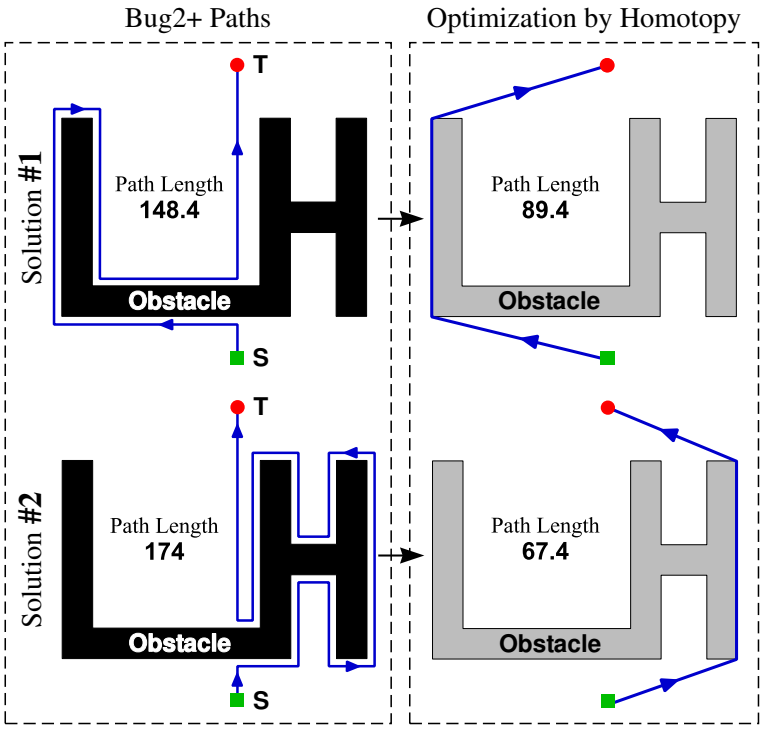


Figure 6.4: An example where the worst Bug2+ path (#2) turns into the optimal solution after optimization.

6.2.2.2 Accelerated Mode of Operation of ABUG

$A^\star$-based anytime approaches make frequent use of the fact that, in many domains, inflating the heuristic values often results in a substantial speed-up at the cost of solution optimality (see [79] for some popular examples of this kind of algorithms). Moreover, if the heuristic h employed is consistent, then, by multiplying it by an inflation factor $\epsilon > 1$, the strategy produces a solution which is ensured not to cost more than ϵ times the cost of the optimal path.

The previous idea can be exploited to provide *ABUG* with an accelerated mode of operation that additionally gives bounds on the suboptimality of the solutions generated. To this end, a simple change is required in line 1 of algorithm 6.1, which consists in inflating the heuristics by ϵ as discussed before, so that the final expression for the cost function is, therefore, $f(P) = g(P)+\epsilon \cdot h(P)$. Any finite real value larger than or equal to one can be assigned to the inflation factor ϵ, which constitutes the first and only user-definable parameter of our approach. By setting $\epsilon = 1$, the strategy *ABUG* adopts its original and non-inflated form. In such a case, a series of paths $\Pi = \{\pi_1, \ldots, \pi_q\}$ increasingly sorted by length is progressively found and improved. On the other hand, for $\epsilon > 1$, the same Π paths as before are computed but the sequence in which they are obtained does not apparently obey to any ordering (in section 6.2.2.3 —fourth property—, some restrictions will be imposed with regard to such a sequencing).

6.2.2.3 Theoretical Properties of ABUG

The most important theoretical properties of *ABUG* are enumerated next. Some additional notation is, nevertheless, introduced first.

Notation

As was already said, $\Pi = \{\pi_1, \ldots, \pi_q\}$ represents the set of non-optimized solutions found by *ABUG* to the path-planning problem at hand. The approach improves each solution $\pi \in \Pi$ by computing the shortest path for the homotopy class to which π belongs. Consequently, let $\Pi^* = \{\pi_1^*, \ldots, \pi_q^*\}$ be the resultant set of improved solutions ($\pi_k^* = \text{SHP}(\pi_k)\ \forall \pi_k \in \Pi$, where SHP refers to the shortest homotopic path function). On the other hand, the cost —Euclidean length— of paths $\pi_k \in \Pi$ and $\pi_k^* \in \Pi^*$ is given by, respectively, σ_k and σ_k^*. This results in two new sets: $\sigma = \{\sigma_1, \ldots, \sigma_q\}$ and $\sigma^* = \{\sigma_1^*, \ldots, \sigma_q^*\}$. Additionally, σ_{best} and σ_{best}^* are used to designate the length of the shortest paths in Π and Π^* (or in other words, $\sigma_{best} = min\{\sigma\}$ and $\sigma_{best}^* = min\{\sigma^*\}$). Following the same terminology, $\sigma_{best,k}$ denotes the minimum cost for a subset of the solutions of Π. More precisely, $\sigma_{best,k} = min\{\sigma_l \in \sigma \mid l \geq k\}$ (observe that $\sigma_{best,1} = \sigma_{best}$ as a particular case of the formulation). Finally, to conclude, π_{opt} symbolizes the optimal solution to the path-planning problem, and σ_{opt} its cost / length.

Properties

*p*1. $\forall \pi_k, \pi_l \in \Pi$ such that $k \neq l$, $\pi_k \neq \pi_l$.

*p*2. $\forall \sigma_k \in \sigma$, σ_k is bounded by expression 6.1.

*p*3. The maximum number of paths found by *ABUG* never goes above the limit

$$| \Pi | \leq 2^{\frac{n}{2}} \tag{6.2}$$

where n denotes the number of intersections between the m-line and the boundary of the obstacles in the environment $\left(\text{i.e. } n = \sum_i n_i\right)$.

*p*4. When performing *ABUG* with an inflation factor $\epsilon = 1$, $\sigma = \{\sigma_1, \ldots, \sigma_q\}$ becomes a totally ordered set under the relation $\leq$ ($\sigma_1 \leq \sigma_2 \leq \ldots \leq \sigma_q$). On the other hand, if $\epsilon > 1$, the following holds: assuming that the execution of the algorithm *ABUG* is in a state where k paths $\pi_1, \ldots, \pi_k$ have been computed[6], the length of the next path to be obtained π_{k+1} is bounded by inequality 6.3. As can be observed, the strategy produces a solution which is guaranteed not to have an additional cost on the best of the paths that remain to be found ($\sigma_{best,k+1}$) of more than $\epsilon - 1$ times the cost of moving from S to T by following a straight-line path ($d(S,T)$). In this way, we can provide bounds on the suboptimality of the solutions generated by *ABUG* when applying the accelerated mode of operation described in section 6.2.2.2.

$$\sigma_{best,k+1} \leq \sigma_{k+1} \leq \sigma_{best,k+1} + (\epsilon - 1) \cdot d(S,T) \tag{6.3}$$

*p*5. Some scenarios, such as the one of figure 6.5(a), can be constructed where σ_{best}^* and σ_{opt} do not match each other ($\sigma_{best}^* > \sigma_{opt}$), which means that the strategy *ABUG* does not always end up yielding the optimal path π_{opt}. However, there is a particular class of problems where π_{opt} is guaranteed to be in the set of —improved— solutions computed by our approach ($\sigma_{best}^* = \sigma_{opt}$). First of all, assume an environment composed of obstacles of generic shape meeting the requirements imposed by the Jordan Curve Theorem [72] (the contour of each obstacle ∂O_i defines a simple closed curve). On

[6] $k \in \{0, \ldots, q-1\}$. In case $k = 0$, the set $\{\pi_1, \ldots, \pi_k\}$ is supposed to be empty

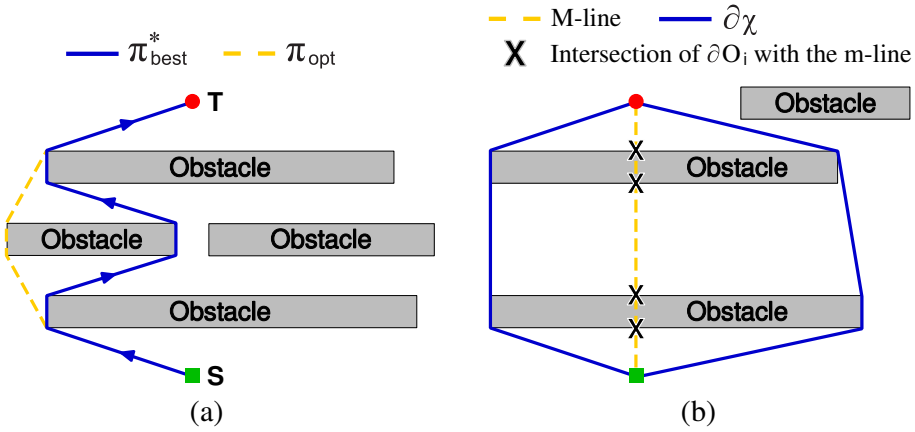


Figure 6.5: About the optimality of our planner: (a) comparing the best solution computed by *ABUG* (π^*_{best}) with the global optimal path (π_{opt}); (b) illustration of the conditions required for optimality.

the other hand, let χ denote the minimal convex subset of $\mathbb{R}^2$ containing S, T, and the contour curve points of those obstacles which intersect the m-line in only two locations (i.e. $\partial\chi = H_{convex}\Big(\{\mathrm{S},\,\mathrm{T}\} \bigcup \big(\bigcup_i \partial O_i \mid n_i = 2\big)\Big)$, where ∂ means *the boundary of* and H_{convex} represents the geometric concept of convex hull). Then, if equation 6.4 holds —see figure 6.5(b) for a case where it happens—, the algorithm *ABUG* can be formally proved to find π_{opt}, or in other words, the set of solutions given by the strategy converges to the optimal value when having enough time for deliberation.

$$\chi \bigcap \left(\bigcup_i O_i \quad | \quad n_i \neq 2 \right) \;=\; \emptyset \tag{6.4}$$

6.2.3 Experimental Results

This section compares *ABUG* with other competing approaches. Some of the most popular path-planning techniques have been included into the comparison by choosing from each a representative member. More precisely, the planners considered are: *NF1* [80], *ARA** [74][7], and *RRT* [81][8]. The planner *NF1* represents the simplest and more efficient way of building an artificial potential function with its only minimum located at the target point T. By applying a wave-propagation technique and a gradient-descent method, *NF1* computes the shortest collision-free path from S to T[9]. On the other hand, *ARA** is a heuristic-based

[7] In [74], the strategy *ARA** is favorably compared against another anytime algorithm named *Anytime A** [73]. Consequently, if the comparative study of this section demonstrates that *ABUG* is clearly more efficient than *ARA**, we can expect *ABUG* to outperform Anytime A* as well

[8] By including the strategy *RRT* into the comparative study, we are also considering, although only in part, the anytime version of such an algorithm named *ARRT* [75]. The *Anytime* RRT approach computes its first path / solution by growing a standard *RRT* without any cost considerations

[9] A faster version of the navigation function *NF1* can be found in [39], where the propagation of the wave was restricted to a small rectangular region containing both the current robot's position (S) and the target (T). Such a region was progressively widened until the search did supply a solution. It seems obvious that avoiding to compute the navigation function for the entire space significantly reduces the computational cost of the planner. Nevertheless —and it was not mentioned by the authors—, the resultant strategy is not optimal

anytime strategy which operates by executing a series of $A^\star$ searches with decreasing inflated heuristics. The approach provides suboptimality bounds for each successive search whose solution is guaranteed not to cost more than ϵ times the cost of the optimal path. $ARA^\star$ gains efficiency by making each $A^\star$ search reuse the results of the previous search iterations. Finally, regarding probabilistic / sampling-based planners, a goal-biased version of the algorithm *RRT* has also been taken into account. This randomized strategy incrementally builds a search tree that attempts to rapidly and uniformly explore the free space. *RRT* has shown to be extremely good in finding feasible paths but with no control on the quality of the solutions produced.

Four different scenarios are proposed to assess the performance of *ABUG* against each of the above-mentioned planners (see figures 6.6 and 6.7(a)). The first mission is intended to test the ability of the strategies to realize that the path-planning problem has no solution. It is important to note that sampling-based methods —and, in particular, the *RRT* planner— do not assume that such a situation can happen because they were essentially conceived to be applied to high-dimensional path-planning problems where the complete exploration of the search space is not possible in a reasonable time. Considering, however, our work focuses on two-dimensional Euclidean search spaces, an adapted version of the classic *RRT* algorithm capable of becoming aware that the target is not reachable has been used for solving mission 1 (in short, the algorithm recognizes the impossibility when the search tree cannot grow any more). On the other hand, the second mission corresponds to a simple scenario where no obstacles are located in the environment. Beyond this simplicity, mission 3 defines a topologically complex scenario under the form of a multiply-connected maze. Finally, in mission 4, many small obstacles are strategically spread throughout the environment. This last scenario constitutes an important challenge for the algorithm *ABUG* because of the high number of paths / solutions which the strategy finds.

All the scenarios are represented as a grid-based map with a resolution of 5cm, and a size of 150m $\times$ 150m —9,000,000 cells— for mission 3 and 100m $\times$ 100m —4,000,000 cells— for the rest of scenarios.

As for the configuration of parameters, the more usual settings defined by their corresponding authors have been used for the strategies *NF1*, goal-biased *RRT*, and $ARA^\star$. More precisely, regarding the latter, the inflation factor has been set to $\epsilon = 3.0$, decreasing in 0.5 steps, which leads to a succession of five $A^\star$ searches. On the other hand, the algorithm *ABUG* has been executed in its default / non-accelerated mode of operation ($\epsilon = 1.0$) in all the scenarios except for mission 4 where the value for ϵ was 3.0 to speed up *ABUG* when dealing with the 1024 resulting paths ($|\Pi| = |\Pi^*| = 512$ in this troublesome mission). Nevertheless, for comparison purposes, the results corresponding to the execution of *ABUG* in its default mode are also reported for mission 4.

Figures 6.6 and 6.7(a) present the results obtained on the four scenarios previously described. The processing times provided correspond to a PC laptop Intel Core Duo @ 1.66 GHz running Windows XP Media Center SP2. Observe that, in mission 2, just one solution — and not five— is given for the strategy $ARA^\star$. This is because the path found with $\epsilon = 3.0$ —the first and highest inflation factor for $ARA^\star$ according to the proposed parameter setting— was already optimal being thus irrelevant the four remaining solutions.

The experimentation of figures 6.6 and 6.7(a) can be summarized as follows: in mission 1, *ABUG* was 57 times faster than the best competing approach in detecting the impossibility

as opposite to the original approach. Despite this problem, which is shared with *ABUG* in the general case, the results reported in section 6.2.3 regarding *NF1* do really come from the speeded-up version of the algorithm

of reaching the target; in the rest of scenarios, *ABUG*, on average, computed its first path 10 times more rapidly and converged to the optimal solution 38 times quicker than the other approaches.

To conclude, figure 6.7(b) depicts the performance of the algorithms when executed on a microcontroller (32-bit Motorola MC68332 @ 25 MHz), which is the typical computational resource that is found in the majority of low-cost robots. The path-planning problem consisted in solving mission 4 again but with a lower resolution than before, namely 50cm cells or 40,000 cells. Concerning the anytime approaches, only the time needed for providing the first path is given in figure 6.7(b). This is essentially due to space reasons as well as to the high-computational cost of $ARA^\star$.

According to the results of figure 6.7(b), it seems clear that the only algorithm with a reasonable response time for the involved planning task is *ABUG*.

6.2.4 Conclusions

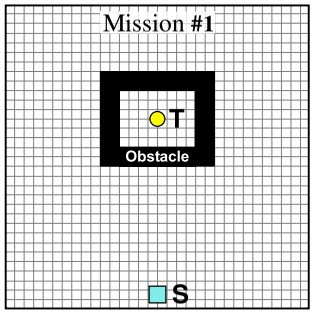
A deterministic anytime path planner inspired by a *Bug*-derivative algorithm has been formally described and, later, compared against some well-known strategies in the field, such as *NF1*, $ARA^\star$ (indirectly, *Anytime* $A^\star$ as well), and *RRT* (indirectly, *ARRT* as well). The performance of the approach proposed, *ABUG*, is significantly better than the one provided by the aforementioned competing planners. *ABUG* efficiently computes a succession of progressively better solutions —or rapidly indicates failure when the given target is unreachable— for each of the path-planning problems considered. On the other hand, *ABUG* makes planning on low-cost robots feasible since the strategy is able to accomplish planning tasks in a reasonable time even in troublesome scenarios such as the one of mission 4.

ABUG has been defined for two-dimensional Euclidean configuration spaces. However, the strategy can be extended to higher-dimensional problems maintaining both the efficiency of the algorithm and some of its more relevant properties, although at the cost of not guaranteeing convergence to the optimal solution. Such an extension, assuming a three-dimensional configuration space, could be achieved by searching for *Bug*-compliant paths in two-dimensional manifolds containing, each of them, the initial and the target configurations (at the time of this writing, *ABUG* has already been extended as indicated above; as an example evidencing this extension, figure 6.7(c) shows a path computed by *ABUG* in a three-dimensional grid-based environment).

6.3 *vABUG*: A Fast Anytime Path Planner Inspired in the Biological Behavior of Insects with Visual Sensing

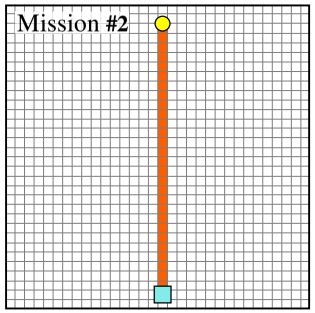
This section extends the work presented in section 6.2 by describing an anytime path planner based on another Bug-derivative algorithm. Such a difference provides the new strategy, named *vABUG*, with enhanced control over the quality / length of the solutions / paths that are computed. Consequently, keeping in mind the close relationship between the time required for calculating a solution and its corresponding quality, it seems clear that *vABUG* adapts better than *ABUG* to the criticality of the planning task by being able to produce an initial set of longer / worse paths when reducing the available time for deliberation —or do the opposite in case the time restrictions are not so severe.

The rest of this section is organized as follows: section 6.3.1 discusses about a new Bug-like strategy called *VisBug+*, while section 6.3.2 shows how VisBug+ can be used to construct



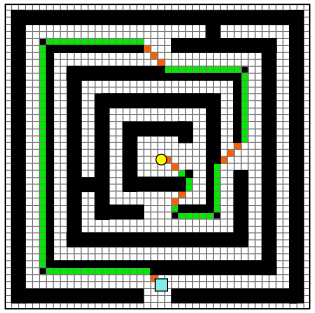
Alg.	t_U
NF1	689
RRT	1,572,401
ARA*	313,624
ABUG ($\epsilon = 1$)	12

(a)



Alg.	t_1	$\frac{l_1}{\sigma_{opt}}$
NF1	581	1.00
RRT	94	1.25
ARA*	1,437	1.00
ABUG ($\epsilon = 1$)	6	1.00

(b)



Alg.	t_1	$\frac{l_1}{\sigma_{opt}}$	t_2	$\frac{l_2}{\sigma_{opt}}$	t_3	$\frac{l_3}{\sigma_{opt}}$	t_4	$\frac{l_4}{\sigma_{opt}}$	t_5	$\frac{l_5}{\sigma_{opt}}$
NF1	28,765	1.00								
RRT	706	1.23								
ARA*	127 $\times 10^3$	1.03	128 $\times 10^3$	1.03	128 $\times 10^3$	1.03	264 $\times 10^3$	1.01	357 $\times 10^3$	1.00
ABUG ($\epsilon = 1$)	94	1.55	578	1.41	1,657	1.00				

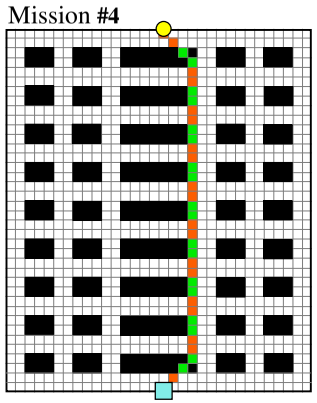
(c)

$\boldsymbol{t_U}$ = time elapsed (ms) until reporting that the goal is unreachable

$\boldsymbol{l_i}$ = length of the ithpath

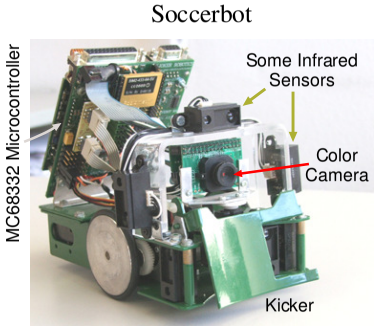
$\boldsymbol{t_i}$ = time instant (ms) in which a strategy provides its ithpath (remember that, in case of *ABUG*, a solution is not provided for each element in the sets Π and Π^* but after finding / computing a path which is shorter than all the previous ones —worse paths are simply rejected)

Figure 6.6: Experimental set-up and comparative results for missions 1 to 3: (a, b, c) processing times and length of the paths in the order that they were produced by the planners *NF1*, *RRT*, *ARA**, and *ABUG* (as a side note, the shortest path found by *ABUG* (π^*_{best}), if exists, is superimposed on the layout of each mission; observe also that *ABUG* assumes that navigation is allowed along the obstacle boundaries).



Alg.	t_1	$\frac{l_1}{\sigma_{opt}}$	t_2	$\frac{l_2}{\sigma_{opt}}$	t_3	$\frac{l_3}{\sigma_{opt}}$	t_4	$\frac{l_4}{\sigma_{opt}}$	t_5	$\frac{l_5}{\sigma_{opt}}$
NF1	2,148	1.00								
RRT	153	1.73								
ARA*	34 $\times 10^3$	1.18	43 $\times 10^3$	1.15	44 $\times 10^3$	1.14	73 $\times 10^3$	1.09	195 $\times 10^3$	1.00
ABUG ($\epsilon = 1$)	1,093	3.51	1,097	1.00						
ABUG ($\epsilon = 3$)	23	3.63	37	1.86	46	1.56	60	1.02	3,542	1.00

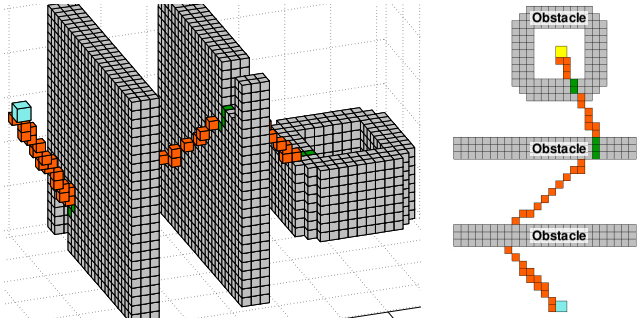
(a)



Mission #4

Alg.	t_1
NF1	26,060
RRT	23,730
ARA*	926,590
ABUG ($\epsilon = 3$)	2,770

(b)



(c)

l_i = length of the ithpath

t_i = time instant (ms) in which a strategy provides its ithpath (remember that, in case of *ABUG*, a solution is not provided for each element in the sets Π and Π^* but after finding / computing a path which is shorter than all the previous ones —worse paths are simply rejected)

Figure 6.7: Experimental set-up and comparative results for mission 4: (a) processing times and length of the paths in the order that they were produced by the planners *NF1*, *RRT*, *ARA**, and *ABUG*; (b) time required to obtain the first path when the planners are executed on a 32-bit Motorola MC68332 @ 25 MHz microcontroller (notice that, in (a), *NF1*, *RRT*, *ARA**, and *ABUG* were run on a PC laptop Intel Core Duo @ 1.66 GHz). Additionally, as a part of this figure, (c) shows that *ABUG* scales to three-dimensional path planning problems.

the fast anytime global path planner known as *vABUG* (besides, this section formally analyzes the properties of *vABUG*); section 6.3.3 reports some experimental results of our algorithm, and compares these results with those obtained by some techniques that are well-known in the field of anytime path planning; and, lastly, section 6.3.4 draws some conclusions.

6.3.1 The Algorithm *VisBug+*

The anytime approach *vABUG* plans paths by using an improved version of an algorithm named *VisBug* [55]. This enhanced version, called *VisBug+*, has several advantages with respect to the original approach. Among all these advantages, the most important one is that VisBug+ produces shorter paths —or, in the worst case, the same path as VisBug— without an increase in complexity.

Next, the key differences between VisBug and VisBug+ are highlighted, with a special emphasis on what makes VisBug+ achieve a better path length performance.

6.3.1.1 Assumptions

The planner VisBug+ makes the following assumptions about the robot and the environment:

- The mobile robot is considered to be a point equipped with a vision sensor, which mimics a typical range finder in the sense that it provides the vehicle with the coordinates of those obstacle boundary points lying within a limited field of view around the robot. On the other hand, the problem of localization is supposed to be solved so that the vehicle can know its current position and the one of the target. Finally, the robot is capable of moving everywhere in free space as well as along the contour of obstacles.
- As for the navigation environment, it is assumed to be static, unknown, and two-dimensional.

6.3.1.2 Some Definitions and Notation

S, T, and C denote, respectively, the starting point of the mission, the target, and the current location of the robot. The line joining S and T is referred to as *main line*, or *m-line* in short. The field of view associated with the robot's range finder is determined by a disc of radius r_v centered at C. Additionally, a point Q is said to be *visible* by the robot if both it is located within its field of view and the straight-line segment with endpoints C and Q does not cross any obstacle. On the other hand, a point Q is *contiguous* to another point U over the set $\{P\}$, if Q can be continuously connected with U using only points of $\{P\}$. The term *contiguous set of visible points* combines the two previous definitions and means that every point in the set $\{P\}$ is visible as well as any pair of points are contiguos to each other over $\{P\}$. Finally, O_j symbolizes a certain obstacle of the environment and ∂O_j its contour curve.

6.3.1.3 Basic Description

The strategy VisBug+ starts by placing the robot at S, i.e. $C = S$. Afterwards, two different processes are sequentially applied: the first one takes care of defining a local / intermediate target point (T_i), while the second process moves the robot one step in the direction of T_i. The execution of the above-mentioned processes is repeated until either $C = T_i = T$ — meaning the successful completion of the path-planning task— or the algorithm does realize the target is not reachable.

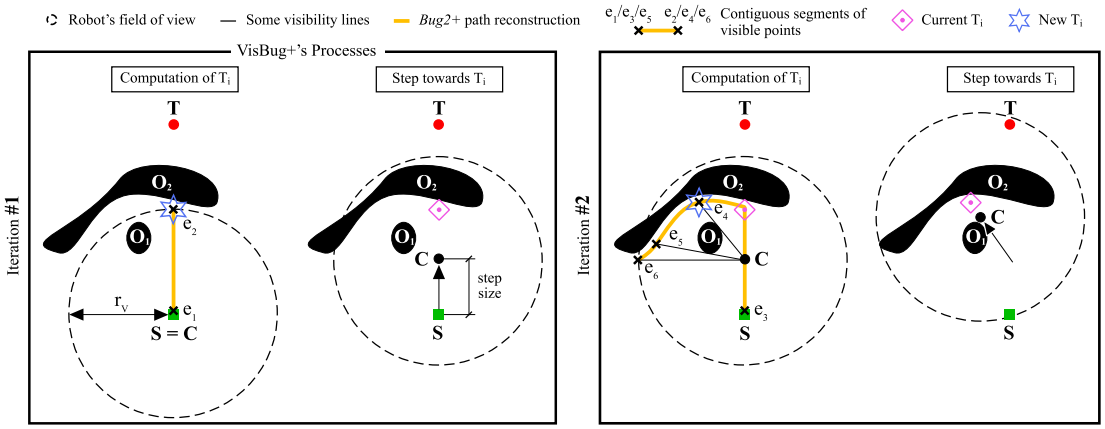


Figure 6.8: The algorithm VisBug+ step by step. Due to the reduced number of iterations considered, a big step to T_i is assumed. Notice that the less the *step size*, the smoother the planned trajectory and the larger the number of iterations needed by VisBug+ to complete the path-planning task. On the other hand, the length of the resultant path is influenced by the size of the robot's field of view or, in other words, by the value assigned to the parameter r_v.

In the way from S to T, local / intermediate targets are generated as follows. First of all, VisBug+ reconstructs, within the current field of view, the path which would be produced by the Bug-like strategy *Bug2+* (see section 6.2.1). Later, from such a path, contiguous sets of visible points are detected and, finally, the farthest point of the contiguous set containing the last-defined intermediate target becomes the new T_i. By way of example, figure 6.8 illustrates the two first iterations of the algorithm in a scenario with several obstacles. As can be observed, while trying to update T_i in the second iteration, two visible contiguous segments of the Bug2+ path are found: e_3e_4 and e_5e_6. As the current T_i is part of segment e_3e_4, the endpoint e_4 turns into the new intermediate target.

It is important to note that, in VisBug+, the task requiring the highest computational cost is the detection of all contiguous sets. However, this task can be significantly simplified by realizing that the computation of T_i is always based on the contiguous set which determined such an intermediate target in the previous iteration of the algorithm —for the first iteration, T_i is chosen from the visible Bug2+ path segment starting at S. Let us next use figure 6.8 as an example of the preceding fact. As can be guessed from the figure, the calculation of T_i in the second iteration comes essentially down to the visibility-based enlargement, in accordance with the up-to-date location of the robot (C), of the contiguous set e_1e_2, not being really necessary the detection of other sets such as e_5e_6. Additionally, observe that the contiguous set e_1e_2 is the one that was employed for deciding the new coordinates of T_i in the first / previous VisBug+ iteration. As a final step after extending e_1e_2, T_i is moved to the endpoint of the resultant contiguous set (e_3e_4) that involves the greatest progress along the Bug2+ path. This simplicity for the process which provides a new intermediate target —the most computationally-intensive part of the algorithm— makes the strategy VisBug+ much faster than others well-known Bug-like approaches that also exploit range data such as *DistBug* [56] and *TangentBug* [57]. On the other hand, the main disadvantage of VisBug+, with respect to these popular approaches, is its generally worse path length performance. Nevertheless, recall that, in an anytime context, there is no point in calculating a high-quality solution to the path-planning problem at hand if this solution is obtained once the available time for the planner has run out.

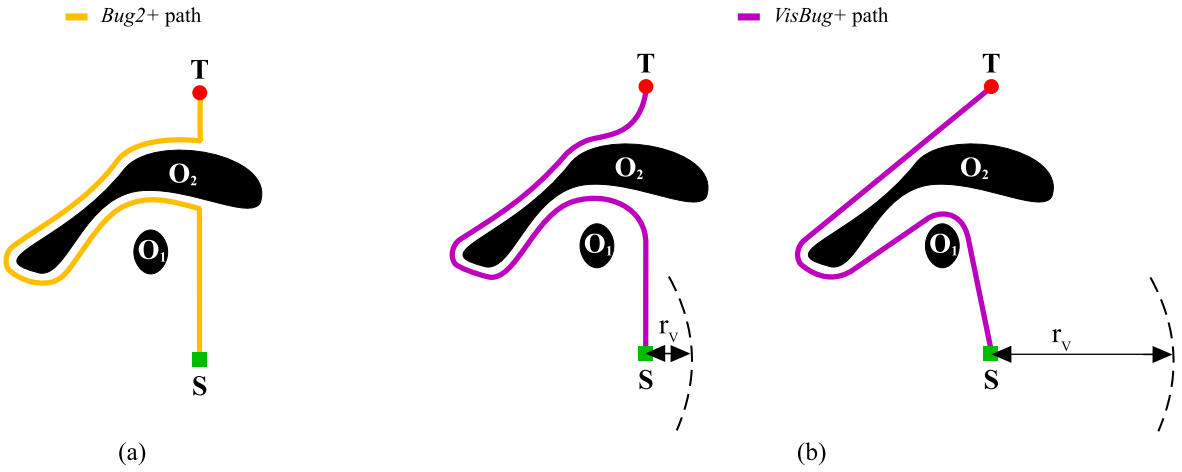


Figure 6.9: Illustration of the fact that the algorithm VisBug+ provides a shortcut of the path produced by the algorithm Bug2+: (a) the Bug2+ path; (b) the path of (a) after being shortcut by VisBug+ (as can be seen, this chart depicts the paths computed by VisBug+ under two different parameterizations, which essentially differ in the value given to the r_v parameter; as for the *step size* parameter, it is assumed to be very small).

Summarizing the previous description, VisBug+ can be understood as a strategy that finds shortcuts on the path generated by Bug2+ (this shortcutting behavior of VisBug+ is readily appreciated in figure 6.9 by comparing the whole paths that would be produced by Bug2+ and VisBug+ in the scenario of figure 6.8). Relating the Bug2+ approach, it is important to recall that it was put forward in chapter 4 and constitutes an improvement of the planner *Bug2* [53][10], in the sense that Bug2+ does provide a path whose length is always shorter than or equal to that of Bug2 (look at figure 6.1 for an example where Bug2+ outperforms Bug2).

To conclude, notice that the main point of discrepancy between the strategies VisBug+ and VisBug is the path from which shortcuts are found: an enhanced Bug2-based path for the former coming from algorithm Bug2+, and the typical Bug2-based path for the latter. As was mentioned before, Bug2+ is a complete planner that never supplies a path longer than Bug2 —its most-directly competing approach—, which implies that the strategy VisBug+ uses a never-worse and usually better reference path for producing shortcuts than VisBug. As was widely discussed in chapter 4 —and is evidenced in figures 4.17 and 6.1—, by executing Bug2+, we obtained the path that would be generated according to Bug2 but with no cycles, if any. The length of these cycles after shortcutting represents the additional cost that a path planned by the strategy VisBug would have with respect to VisBug+. As a final comment, it is worth to remark that the length of any path provided by Bug2+ —and, consequently, by VisBug+— does not exceed the upper bound given by expression 6.1.

[10] see also section 4.1

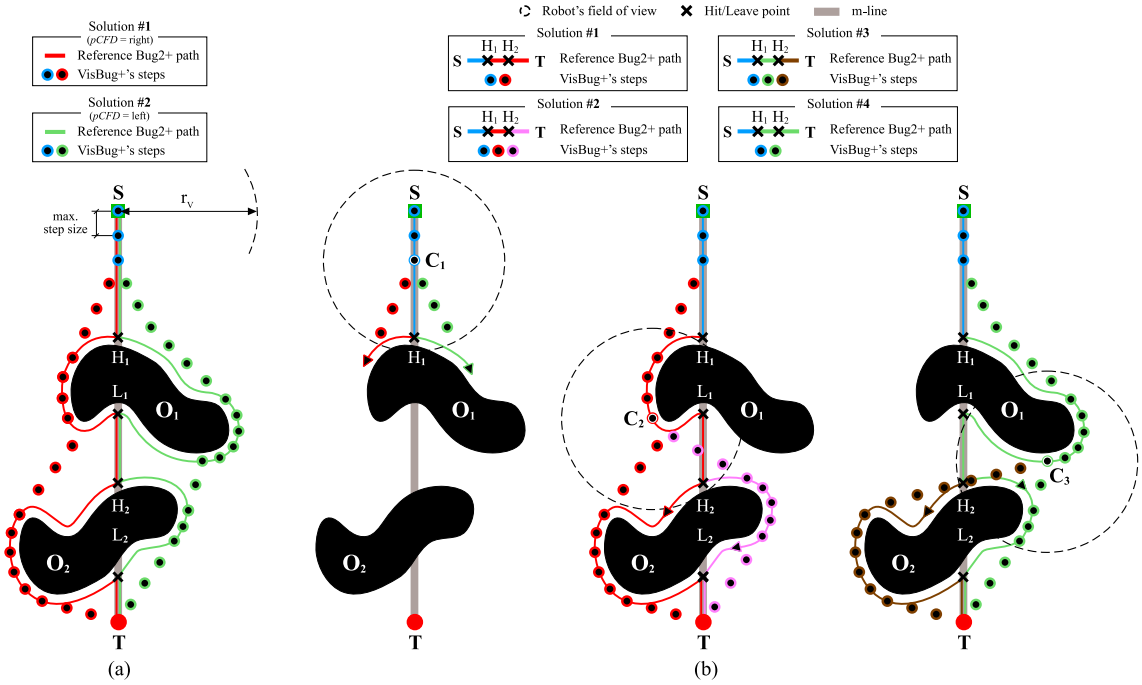


Figure 6.10: Fundamentals of the strategy *vABUG*: (a) running VisBug+ with different settings for the parameter *pCFD*; (b) multiple path search performed by vABUG.

6.3.2 The Algorithm *vABUG*

The algorithm VisBug+, as described in section 6.3.1, allows us to plan a single path in an unknown and static environment. The particular features of such a path depend on how the parameters *step size*, r_v, and *pCFD* are set by the user. Among these VisBug+'s parameters, *pCFD* had not been really mentioned so far. To fill this gap, notice that *pCFD* does determine the direction —left or right— in which the robot will go around the contour of the obstacles. By way of example, figure 6.10(a) presents the two solutions that would be obtained by executing VisBug+ with *pCFD* = *left* and *pCFD* = *right* in a simple scenario.

From figure 6.10(a), it is clear that different paths can be obtained with VisBug+ by choosing distinct obstacle contour following directions, or in other words, by varying the value given to the *pCFD* parameter. The anytime approach *vABUG* exploits this fact to produce multiple paths in an a priori known environment. In this respect, however, it is important to note that *vABUG* is not limited to merely execute the planner VisBug+ twice by assigning *left* and *right* to *pCFD*, but the strategy goes beyond by generating a wider set of topologically different solutions, which correspond to the distinct ways of circumnavigating the obstacles that are located between S and T. Figure 6.10(b) illustrates this exhaustive search in a mission where four paths were successfully planned. As will be seen later, within the set of solutions provided by *vABUG*, there is, in most cases, one that is, from a topological point of view, equivalent to the global optimal path. This circumstance is taken into account in a final stage of the algorithm to make *vABUG* converge to the optimal solution whenever possible.

Next, the strategy *vABUG* is described in more detail emphasizing its most relevant theoretical properties.

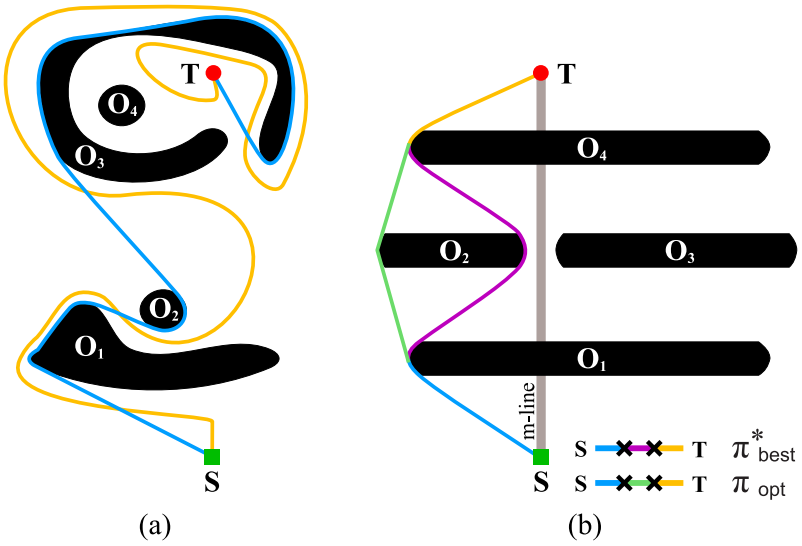


Figure 6.11: (a) in blue, the shortest path of the homotopy class associated with the yellow trajectory; (b) a scenario in which vABUG does not converge to the global optimal path.

6.3.2.1 Description of the Planner

The planner *vABUG* searches for paths by paying special attention to the event associated with the detection of new obstacles. The triggering of such an event means that an obstacle is currently impeding the progress of the robot to T and, consequently, that the obstacle has to be circumnavigated. *vABUG* explores all possibilities of avoiding an obstacle, which are reduced to just two in $\mathbb{R}^2$: from the branch point, follow the contour of the obstacle in the left and right directions. With this purpose, *vABUG* launches an initial search based on the strategy VisBug+ and, later, waits for the above-mentioned event to occur. Once an obstacle is found, the search is widened by defining two instances of VisBug+: one with $pCFD = left$ and the other with $pCFD = right$. The wait and widening steps are repeated until all the VisBug+-type processes involved into the search provide a solution to the path-planning problem (or indicate failure if no solution can be obtained). This way of acting for the algorithm *vABUG* is faithfully reflected in figure 6.10(b). Remember that the robot is assumed to be equipped with a vision sensor so that it is able to sense obstacles at some distance from them —r_v, at the most— as can be observed in figure 6.10(b), where the points denoted as C_i indicate the robot's locations in which a new obstacle was detected, or in other words, in which the widening step was executed. From the viewpoint of a VisBug+-type process, each C_i corresponds to the definition of a hit point over the part of the reference Bug2+ path that lies within the robot's field of view.

Once all the VisBug+-compliant solutions have been generated, the algorithm *vABUG* goes to a second stage where the quality of such solutions are intended to be improved while time permits. To this end, *vABUG* makes use of the topological concept of *path homotopy*, which provides us with an efficient way for optimizing a given path by solving the so-called *shortest homotopic path* problem [72]. Informally speaking, a path is regarded as an elastic band joining the points S and T which is tightened to shorten it (see figure 6.11(a) for an example). Among the many methods that have been proposed to compute the shortest homotopic path in $\mathbb{R}^2$, *vABUG* makes use of the one published in [78] because it exhibits the lowest algorithmic complexity, to the best of our knowledge.

Finally, observe that, *vABUG* produces a new solution each time a path is either computed —first stage— or optimized —second stage—, and it turns out to be shorter (better) than the best path found so far. In this way, the strategy guarantees the generation of a strictly monotonically decreasing sequence of solutions regarding path length.

6.3.2.2 Theoretical Properties

The most important theoretical properties of *vABUG* are enumerated next after the introduction of some additional notation.

Notation

$\Pi = \{\pi_1, \ldots, \pi_q\}$ represents the set of solutions found by *vABUG* in its first stage on the basis of the algorithm VisBug+ (notice that the indexes of the paths define the order in which they were obtained). As already known, the strategy *vABUG* improves each solution $\pi \in \Pi$ by computing the shortest path for the homotopy class to which π belongs. Consequently, let $\Pi^* = \{\pi_1^*, \ldots, \pi_q^*\}$ be the resultant set of optimized solutions ($\pi_l^* = \text{SHP}(\pi_l)\ \forall \pi_l \in \Pi$, where SHP refers to the shortest homotopic path function). On the other hand, the cost —Euclidean length— of the paths $\pi_l \in \Pi$ and $\pi_l^* \in \Pi^*$ is, respectively, σ_l and σ_l^*. This results in two new sets: $\sigma = \{\sigma_1, \ldots, \sigma_q\}$ and $\sigma^* = \{\sigma_1^*, \ldots, \sigma_q^*\}$. Additionally, σ_{best}^* is used to designate the length of the shortest path in Π^* (or in other words, $\sigma_{best}^* = min\{\sigma^*\}$). Finally, to conclude, π_{opt} symbolizes the global optimal solution to the path-planning problem, and σ_{opt} its cost / length.

Properties

*p*1. $\Pi = \Pi^* = \emptyset$ if and only if T is not reachable.

*p*2. $\forall \pi_l, \pi_m \in \Pi$ such that $l \neq m$, $\pi_l \neq \pi_m$.

*p*3. $\forall \sigma_l \in \sigma$, σ_l is bounded by expression 6.1. Moreover, $\forall \sigma_l \in \sigma$, $\sigma_l^* \in \sigma^*$ it holds that $\sigma_l \geq \sigma_l^*$.

*p*4. The maximum number of paths generated by the strategy *vABUG* never goes above the limit

$$| \Pi | = | \Pi^* | \leq 2^{\frac{n}{2}} \tag{6.5}$$

where n designates the number of intersections between the m-line and the boundary of the obstacles (i.e. $n = \sum_j n_j$).

As can be guessed, the maximum number of solutions provided by *vABUG* changes in accordance with the complexity of the environment under consideration: in short, the higher the number of obstacles between S and T, the higher the number of ways of circumnavigating them and, consequently, the higher the cardinality of the Π/Π^* sets. This is a very interesting property not found in any other anytime approach —to the best of our knowledge— where the desired number of solutions to be obtained must be decided a priori (see e.g. *ARA** [74] and *ARRT* [75]). However, the only relevant parameter in *vABUG* is r_v which, from a practical point of view, allows trading off running time and quality for the solutions of the Π set.

*p*5. Some scenarios can be constructed where σ^*_{best} and σ_{opt} do not match each other ($\sigma^*_{best} > \sigma_{opt}$), which implies that the strategy *vABUG* does not always end up yielding the optimal path π_{opt} (see the one in figure 6.11(b)). However, there is a fairly wide class of problems where π_{opt} is guaranteed to be in the set of —optimized— solutions calculated by our approach ($\sigma^*_{best} = \sigma_{opt}$). In order to characterize this class, assume an environment composed of obstacles of generic shape meeting the requirements imposed by the Jordan Curve Theorem [72] (essentially, the contour of each obstacle — ∂O_j— should define a simple closed curve). On the other hand, let χ_j denote the minimal convex set —referred to as *MCS* hereafter— containing ∂O_j (i.e. $\partial\chi_j = H_{convex}(\partial O_j)$, where ∂ means *the boundary of* and H_{convex} represents the geometric concept of convex hull). In addition, let φ be the MCS that includes S, T, and the contour curve points of those obstacles which intersect the m-line, or more formally, $\partial\varphi = H_{convex}\Big(\{S,\ T\} \bigcup \Big(\bigcup_j \partial O_j \ \mid \ n_j \neq 0\Big)\Big)$. Then, if equations 6.6, 6.7, and 6.8 are satisfied, the algorithm *vABUG* can be proved to always find π_{opt} when having enough time for deliberation.

$$\Big(\bigcup_j \chi_j \ \mid \ n_j \neq 0\Big) \bigcap \{S,\ T\} \ = \ \emptyset \tag{6.6}$$

$$\Big(\bigcap_j \chi_j \ \mid \ n_j \neq 0\Big) \ = \ \emptyset \tag{6.7}$$

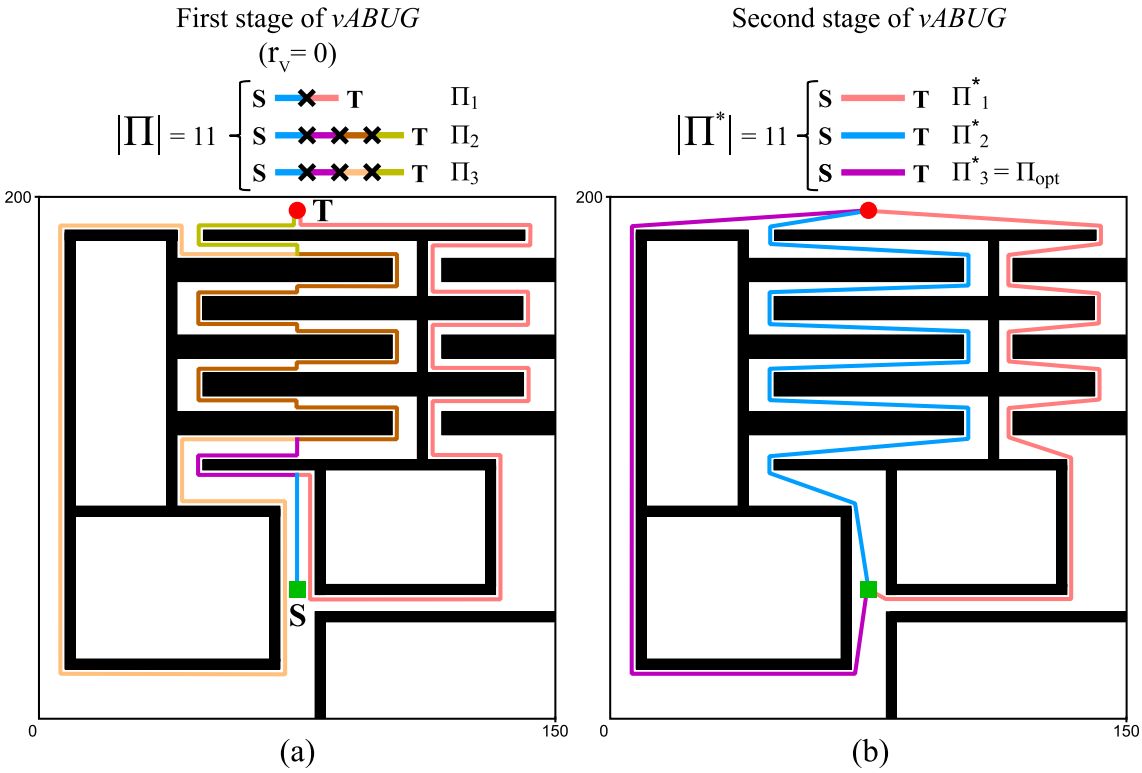
$$\Big(\bigcup_j \chi_j \ \mid \ n_j = 0\Big) \bigcap \varphi \ = \ \emptyset \tag{6.8}$$

6.3.3 Experimental Results and Brief Discussion

This section assesses the performance of the algorithm *vABUG* when carrying out a complex planning task in a two-dimensional configuration space. Besides, *vABUG* is compared against two of the most popular anytime approaches: *ARA** [74] and *ARRT* [75]. The former is a deterministic strategy which operates by executing a succession of A* searches with decreasing inflated heuristics, while the latter corresponds to the anytime version of the probabilistic/sampling-based planner *RRT* (*Rapidly-exploring Random Trees*, see [82]). *ARRT* generates a series of search trees, each producing a solution that is ensured to be less expensive than the previous ones.

Figure 6.12(a) and (b) show the planning task that was intended to be solved. Observe that obstacles are strategically spread through the environment to define three topologically different ways of achieving the target point. On the other hand —and although this is not evident from the figure—, a grid-based representation of the environment was adopted, resulting in a map of 150m $\times$ 200m with a resolution of 5cm (in total, 12000000 cells for the search space).

Figure 6.12(a) presents the first three solutions generated by *vABUG* in the given scenario supposing a zero field of vision for the robot (i.e. $r_v = 0$, which essentially means that obstacles are detected by means of tactile sensing). These three solutions are just a subset of the paths that were really computed in the first stage of the algorithm according to the VisBug+ planner. More precisely, in such a stage, not three but eleven different paths were produced by *vABUG*. However, the eight remaining solutions are omitted because, from a topological point of view, they do not differ from the ones illustrated in figure 6.12(a). Lastly, in the second stage of the algorithm, all the above-mentioned solutions were improved as indicated in section 6.3.2.1.

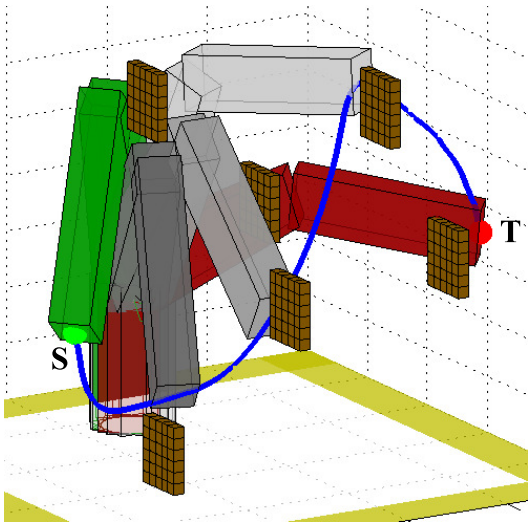


vABUG

		r_V=0	r_V=15	r_V=30	r_V=50
Π_1	l_1	1.54	1.27	1.18	1.09
	t_1	15	27	32	38
Π_2	l_2	1.64	1.52	1.51	1.51
	t_2	15	37	48	61
Π_3	l_3	1.97	1.63	1.43	1.40
	t_3	19	56	79	102
Π^*_1	l_4	1.09			
	t_4	215	697	888	913
Π^*_2	l_5	1.51			
	t_5	407	1249	1452	1477
Π^*_3	l_6	1.00			
	t_6	1715	2569	2952	2977

(c)

	ARA*	ARRT (averaging 100 runs)
l_1	1.56	1.12
t_1	688	375
l_2	1.13	1.09
t_2	35797	531
l_3	1.00	1.06
t_3	58610	703



(d)

Figure 6.12: Some experimentation with vABUG and other anytime path planners.

Figure 6.12(b) depicts the result of applying this optimization/tightening process on the paths of figure 6.12(a). Notice that these three final solutions represent the optimal way, in terms of Euclidean distance, of performing each alternative path to the goal so that the global optimal solution is ensured to be one of them (π_3^*, to be precise).

Figure 6.12(c) summarizes the processing times in msec[11] associated with the strategy *vABUG* (top) as well as *ARA** and *ARRT* (bottom). The figure shows, for each computed path, its normalized length l_i —a value of 1.00 indicates the global optimal path— and the precise time instant t_i at which the corresponding solution was provided. Four sets of results are given for *vABUG*, which come from assigning a different value to the parameter r_v: 0, 15, 30, and 50. As expected, better solutions were obtained as r_v was increased. Specifically, in this particular scenario, the first three solutions of *vABUG* improved their quality in about 22 per cent, on average, for $r_v = 0$ to 50. Besides, the improvement was accomplished with just a little time penalty in comparison with the execution time requirements of both *ARA** and *ARRT*. On the other hand, *vABUG*, at worst, produced its first path 9.87 times faster and converged to the global optimal solution 19.69 times more rapidly than the best competing approach.

To end with, it is important to stress that the algorithm *vABUG* scales well to three-dimensional path-planning problems. By way of example, figure 6.12(d) displays one of the trajectories planned by *vABUG* for a robot arm with three degrees of freedom in an environment involving the avoidance of several obstacles (in the figure, obstacles are represented by the six brown boxes).

6.3.4 Conclusions

A two-stage anytime path planner named *vABUG* which is inspired by a *Bug*-derivative algorithm has been described and, afterwards, successfully compared against some well-known strategies in the field such as *ARA** and *ARRT*. The computational savings provided by *vABUG* make this approach specially suited for planning on low-cost robots typically equipped with simple microcontrollers.

[11]The machine used for testing was a PC laptop Intel Core Duo @ 1.66 GHz running Windows XP SP2

Conclusions and Future Work

7.1 Concluding Remarks

7.1.1 Scope of the Dissertation

This dissertation has addressed the problem of mobile robot navigation from two distinct perspectives, namely *reactive* and *deliberative*.

7.1.1.1 The Task of Navigation as Viewed from the *Reactive* Paradigm

Under the *reactive* paradigm, the task of navigation is intended to be solved by decomposing it into elementary steps. In each step, a new control action is computed for the robot by using the information of the neighboring environment. As it is plain from the latter, robot's actions are fully determined by local knowledge. This fact has both positive and negative implications: on the positive side, notice that, because local knowledge generally involves small amounts of information, non-computationally intensive algorithms are required to plan each robot's action (this ability for planning actions quickly is the cause of the fact that robots that operate reactively are able to be highly-responsive to changes in the environment); on the negative side, it is important to stress that a reactively-controlled robot may fail in navigating from an initial location to a final / target destination, essentially because decisions about actions are not made having a global view of the task to be performed.

7.1.1.2 The Task of Navigation as Viewed from the *Deliberative* Paradigm

Under the *deliberative* paradigm, the task of navigation is conceptually divided into two subtasks: global path planning, and plan execution[1]. On the one hand, *global path planning* is the process of employing a model of the environment for finding the best / near-best sequence of actions that will allow the robot to safely achieve a desired target location —or more strictly speaking, a desired target state. On the other hand, *plan execution* refers to a subtask that is able to both execute and monitor a given sequence of actions.

When a navigation task is solved deliberatively, the process of global path planning is carried out before the robot begins its journey towards the target location. Moreover, once the global path planning has been completed, the output of this process is used to feed the plan execution subtask. In consequence, from this moment forth, such a subtask will move the robot in accordance with the sequence of actions received. During the course of these actions, it is important to highlight that a / an special / anomalous situation may arise. Specifically, this situation corresponds to the case in which the plan execution subtask anticipates[2] that

[1] To be fair, there exists a third subtask called localization

[2] on the basis of the information provided by the robot's sensors

the robot will collide with an obstacle if the sequence of actions suggested by the global path planning process continues being executed (as a side note, this unexpected problem can just occur due to one, or both, of the following reasons: (*r1*) the process of global path planning was based on a model of the environment that was not sufficiently accurate and / or complete; (*r2*) the environment involved dynamic obstacles —i.e. some obstacles in the environment did change their position while the robot was moving towards its target). When a collision is anticipated by the plan execution subtask, a replanning procedure is immediately activated. Concisely, this procedure consists of three steps: (*s1*) build a model that faithfully reflects the observed state of the environment (this new model could be obtained by fusing the information of the available model with the sensor data collected by the robot during navigation); (*s2*) perform the global path planning process using the previously-built model of the environment; (*s3*) order the plan execution subtask to execute the sequence of actions resulting from step *s2*.

As is clear from above, once the robot has started to navigate, the process of global path planning —as a part of the earlier-described replanning procedure— may be required to be executed due to the finding of an obstacle blocking the robot's intended path. When this event takes place, the robot is usually stopped until a new sequence of actions —corresponding to an alternative obstacle-free path to the user-defined target location— is actually computed by the global path planning process. As one can readily imagine, the time the robot will remain stopped is not negligible, since the aforesaid process is deliberative and, hence, inherently time-consuming. In short, the preceding discussion brings to light the well-documented fact that deliberatively-controlled robots can react late to unexpected events.

7.1.2 Summary of the Main Contributions

The major contributions of this dissertation in the fields of reactive navigation and global path planning are separately summarized next.

7.1.2.1 Contributions in the Field of Reactive Navigation

This dissertation has made three relevant contributions to the field of reactive navigation:

- ◇ *C1.* Because of exclusively relying on local sensing, *purely* reactive navigation methods suffer from several limitations. Among these limitations, the most important one refers to the problem of *local minima*. As is well-known, this problem may cause that a purely reactive robot gets stuck indefinitely before reaching its target location.

  In chapter 3, we have proposed two general principles, briefly named T^2, which can be directly applied on a large variety of currently existing purely reactive navigation methods to avoid the trapping situations that are due to the local minima problem. On the basis of the T^2 principles, a robot does acquire the ability for successfully moving out of a local minimum, irrespective of: (1) the precise shape and size of the obstacle/s that is/are causing such a local minimum; and (2) the maximum obstacle detection range of the robot's sensors. (As a clear example of the latter fact, notice that a robot behaving according to T^2 would be able to escape from an extremely deep and wide U-shaped canyon by merely using contact/zero-range sensors). As another key aspect of the T^2 principles, it should also be mentioned that any purely reactive navigation method does keep being reactive after incorporating such principles.

Throughout chapter 3, the T^2 principles have been applied to two existing and very popular purely reactive navigation methods, which are known to be susceptible to the local minima problem. Specifically, the methods chosen for exemplifying the application of T^2 have been the *artificial potential fields* [8] (*PFM*) and the *dynamic window approach* [40, 41] (*DWA*). A T^2-based version of both *PFM* and *DWA* has been described in detail. Furthermore, these versions have been exhaustively tested in simulated and real experiments involving increasingly complex scenarios of navigation, most of them containing intricate obstacles forming local minima. In short, all of the above has permitted to demonstrate the two essential features of T^2: on the one hand, its effectiveness for solving the local minima problem; on the other hand, its generality to be applied to significantly different purely reactive navigation methods, such as *PFM* and *DWA*.

◇ *C2.* As evidenced in section 2.6, there exists a broad family of algorithms for reactive navigation generically called *Bug*. These algorithms stand out because they are proved to guarantee completeness. What is more, they accomplish this by using a very minimal global knowledge of the environment[3].

In the work published in [71], the most popular Bug-like algorithms were compared against each other by means of both the length of the path traversed by the robot, and the computational resources[4] needed to generate such a path. As a result of this comparison, it was concluded that one of the Bug-like algorithms with best trade-off between path length performance and computational resource usage is *Bug2* (see sections 2.6.2 and 4.1 for a detailed description of this specific Bug-like algorithm).

In chapter 4, we have proposed a new version of the algorithm Bug2 named *Bug2+*. As compared to Bug2, this new version does provide a better path length performance, with no additional computational cost. Besides, Bug2+, like Bug2, does ensure completeness.

Finally, it is important to emphasize that, in the last part of chapter 4 —to be precise, in section 4.2.2—, we have proved rigorously all the above-claimed properties of Bug2+.

◇ *C3.* In chapter 5, we have presented the theoretical basis of a new method of reactive navigation called *Bug*T^2. Strictly speaking, this new method emerges as a combination of the T^2 principles (contribution *C1*) and the algorithm Bug2+ (contribution *C2*). As a main benefit of this combination, it should be strongly stressed that *Bug*T^2 does possess the major advantages of T^2 and Bug2+[5]. Specifically, these advantages are: on the one hand, as a result of applying the T^2 principles, *Bug*T^2 exhibits a fairly good path length performance; on the other hand, as inherited from the algorithm Bug2+, *Bug*T^2 is able to ensure, whenever possible, the convergence of the robot to its target.

[3] Notice that the fact of using some global knowledge means that these algorithms perform *non-pure* reactive navigation

[4] essentially, memory consumption and running time

[5] Comparatively speaking, observe that: (1) the T^2 principles, as opposed to the algorithm Bug2+, are not enough to guarantee the completion of any given navigation task; (2) in cases where such principles get to drive the robot to the target location, the resultant path generally compares favorably —in terms of length— to that of the algorithm Bug2+

7.1.2.2 Contributions in the Field of Global Path Planning

This dissertation has made one relevant contribution to the field of global path planning:

- ◇ *C4.* In chapter 6, we have used distinct Bug-like strategies as a basis for developing two deterministic anytime[6] global path planners, named *ABUG* and *vABUG*. To be precise, *ABUG* has been based on our strategy *Bug2+* (contribution *C2*), whereas *vABUG* has been derived from the classical strategy *VisBug* (see section 2.6.3). Both planners have been expressly designed for efficiently finding a series of increasingly better paths in problems of low dimensionality, such as those planning problems typically concerned with low-cost robotics applications. The performance of *ABUG* and *vABUG* has been extensively tested and compared with that of other popular anytime path planners —e.g. *ARA** [75] and *ARRT* [74], just to mention two of them. Under this comparative testing, it has been clearly shown that *ABUG* and *vABUG* do provide better paths much more rapidly than their competitors.

7.1.3 List of Publications

The publications that have been derived from the work presented in this dissertation are enumerated below, categorized by the type of publication.

7.1.3.1 Technical and Research Reports

[**1**] J. Antich and A. Ortiz, "T^2: An approach to robotic navigation in unknown and dynamic environments," Department of Mathematics and Computer Science, University of the Balearic Islands, Tech. Rep. A-3, 2004.

[**2**] J. Antich, "Reactive robotics: A paradigm not limited to simple tasks," May 2006, Research Report, Department of Mathematics and Computer Science, University of the Balearic Islands.

[**3**] J. Antich and A. Ortiz, "A dynamic window approach to navigate in complex scenarios using low-cost sensors for obstacle detection," Department of Mathematics and Computer Science, University of the Balearic Islands, Tech. Rep. A-4, 2007.

[**4**] J. Antich and A. Ortiz, "Bug2+: Details and formal proofs," Department of Mathematics and Computer Science, University of the Balearic Islands, Tech. Rep. A-1, 2009.

7.1.3.2 Refereed Conferences

[**5**] J. Antich and A. Ortiz, "An underwater simulation environment for testing autonomous robot control architectures," in *proceedings of the IFAC conference on Control Applications in Marine Systems*, July 2004, Ancona (Italy), pp. 509–514, ISSN 1474-6670.

[**6**] J. Antich and A. Ortiz, "Extending the potential fields approach to avoid trapping situations," in *proceedings of the IEEE/RSJ international conference on Intelligent Robots and Systems*, August 2005, Edmonton (Canada), pp. 1379–1384, ISBN 0-7803-8913-1.

[6] As a general way of working, an anytime algorithm is able to quickly plan a collision-free suboptimal path from a given starting position to a desired target position. What is more, while the available time for planning is not over, such an initial path is continuously improved

[**7**] J. Antich and A. Ortiz, "Bug-based T^2: A new globally convergent potential field approach to obstacle avoidance," in *proceedings of the IEEE/RSJ international conference on Intelligent Robots and Systems*, October 2006, Beijing (China), pp. 430–435, ISBN 1-4244-0259-X.

[**8**] J. Antich and A. Ortiz, "A convergent dynamic window approach with minimal computational requirements," in *proceedings of the 10th international conference on Intelligent Autonomous Systems*, July 2008, Baden Baden (Germany), pp. 183–192, ISBN 978-1-58603-887-8.

[**9**] J. Antich, A. Ortiz, and J. Mínguez, "ABUG: A fast bug-derivative anytime path planner with provable suboptimality bounds," in *proceedings of the 14th International Conference on Advanced Robotics*, June 2009, Munich (Germany), pp. 1–8, ISBN 978-1-4244-4855-5.

[**10**] J. Antich, A. Ortiz, and J. Mínguez, "A bug-inspired algorithm for efficient anytime path planning," in *proceedings of the IEEE/RSJ international conference on Intelligent Robots and Systems*, October 2009, St. Louis (USA), pp. 5407–5413, ISBN 978-1-4244-3804-4.

[**11**] J. Antich and A. Ortiz, "A rapid anytime path planner with incorporated range sensing to improve control on solution quality," in *proceedings of the 11th international conference on Intelligent Autonomous Systems*, August 2010, Ottawa (Canada), pp. 207–216, ISBN 978-1-60750-612-6.

7.1.3.3 Journals

[**12**] J. Antich and A. Ortiz, "Development of the control architecture of a vision-guided underwater cable tracker," *Intl. Journal of Intelligent Systems*, vol. 20, no. 5, pp. 477–498, 2005, ISSN 0884-8173.

[**13**] J. Antich, A. Ortiz, and G. Oliver, "A PFM-based control architecture for a visually guided underwater cable tracker to achieve navigation in troublesome scenarios," *Journal of Maritime Research*, vol. 2, no. 1, pp. 33–50, 2005, ISSN 1697-4840.

[**14**] J. Antich and A. Ortiz, "Reactive navigation in troublesome environments: T^2 strategies," *Instrumentation Viewpoint*, no. 6, pp. 51–52, 2007, ISSN 1886-4864.

7.1.3.4 Book Chapters

[**15**] J. Antich, A. Ortiz, and G. Oliver, *Reactive Control of a Visually Guided Underwater Cable Tracker.* In book *Robotics and Automation in the Maritime Industries* published by *Instituto de Automática Industrial* (CSIC), 2006, ch. 6, pp. 111–132, IBSN 84-611-3915-1.

[**16**] J. Antich and A. Ortiz, *Traversability and Tenacity: Two New Concepts Improving the Navigation Capabilities of Reactive Control Systems.* In book *Robotics and Automation in the Maritime Industries* published by *Instituto de Automática Industrial* (CSIC), 2006, ch. 7, pp. 133–154, ISBN 84-611-3915-1.

7.2 Forthcoming Work

In the near future, the following tasks are planned to be undertaken:

- ◇ As a natural continuation of the work done in this dissertation, a hybrid navigation system is going to be developed by combining one of the methods that we have put forward for reactive control —i.e. T^2, *Bug2+*, or $BugT^2$— with one of the methods that we have devised for global path planning —i.e. either *ABUG* or *vABUG*. This hybrid system is expected to allow a robot to follow safe and optimal / near-optimal paths in highly dynamic and extremely complex environments.

- ◇ As a second future step, we are going to propose an anytime global path planning method based on the concept of *homotopy* class. The aim of this proposal is to ensure that all paths found by the planner are topologically different (notice that this does not necessarily occur neither in *ABUG* nor in *vABUG*); or in more formal words, the overall objective is that each path returned by the planner belongs to a different class of homotopy (as a side note, two paths in the same homotopy class —i.e. two paths that are homotopic— represent two ways of reaching the target location that wind around obstacles in exactly the same manner).

  Concisely, the new homotopy-based path planner is going to consist of two main sequential stages: the first stage will allow identifying all existing homotopy classes within the search space; afterwards, in the second / final stage, the best path of each previously-identified homotopy class will be computed. As a result of these two stages, a minimally complete set of paths will be planned ('minimal' because no path will be topologically equivalent to any other; and 'complete' because such a set will contain all topologically-different paths that a robot could follow to achieve the desired target).

  At the time of this writing, two versions of the above-explained homotopy-based path planner have already been developed and published in:

  [**17**] E. Hernández, M. Carreras, J. Antich, P. Ridao, and A. Ortiz, "A topologically guided path planner for an AUV using homotopy classes," in *proceedings of the IEEE International Conference on Robotics and Automation*, May 2011, Shanghai (China), pp. 2337–2343, ISBN 978-1-61284-385-8.

  [**18**] E. Hernández, M. Carreras, P. Ridao, J. Antich, and A. Ortiz, "A search-based path planning algorithm with topological constraints. Application to an AUV," in *proceedings of the 18th IEEE/OES IFAC World Congress*, August 2011, Milano (Italy), pp. 13 654–13 659, ISBN 978-3-902661-93-7.

Appendix A

Robots Used for Experimentation

This work has alternatively made use of four different mobile robotic platforms to corroborate, through experimentation, the claimed properties of the algorithms which have been devised. In short, two of these platforms constitute ground-type robots, while the other two are specifically designed for underwater applications. They all are described next.

A.1 Ground Robots

A.1.1 The Robot *Pioneer 3-DX*

Pioneer 3-DX is a general purpose base platform commercialized by *MobileRobots Inc*, which looks as shown in figure A.1. Among its main features, the following are highlighted:

- An aluminum body of 44 cm × 38 cm × 22 cm with two 19 cm differential drive wheels.
- One DC motor per wheel that allows the robot to reach speeds of up to 1.6 metres per second.
- The platform is highly *holonomic*, being able to rotate in place —by moving both wheels at the same speed but in opposite directions— or to swing around a stationary wheel in a circle with a radius of 32 cm.
- A rear caster to balance the robot.
- A maximum payload carrying capacity of 23 kg.
- Sixteen ultrasonic sensors arranged to provide a 360-degree coverage.
- Between 18 and 24 hours of autonomy with fully charged batteries.
- The control commands are sent to the robot via a serial connector through which sonar readings, motor encoder data, and other information are also received.

A.1.2 A Small Robot called *SoccerBot*

SoccerBot is a miniature robot with functionality similar to that of larger robots employed in research and education such as the Pioneer. It has been purposely designed to meet the regulations for RoboCup and FIRA small size leagues. Concisely, these are two initiatives for advanced robotics and AI research around a friendly soccer competition. Figure A.2 depicts the physical aspect of the robot and enumerates, at the same time, its more relevant characteristics which are (refer to [100] for further details):

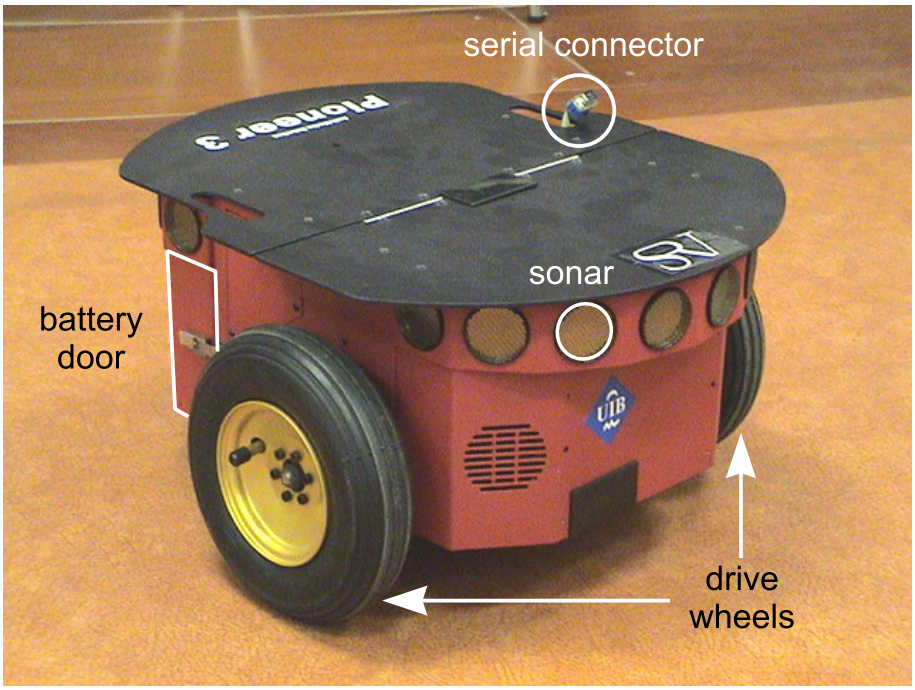


Figure A.1: A Pioneer 3-DX robot.

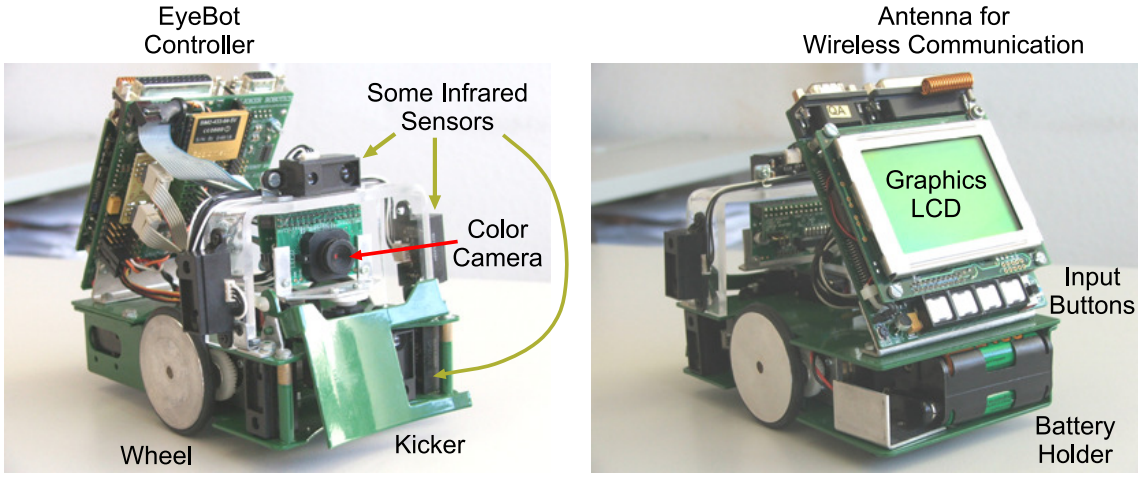


Figure A.2: Our SoccerBot S4X robot distributed by *Joker Robotics*.

- As for the robot dimensions, it fits within an 18 cm circle in diameter.
- SoccerBot has a differential drive actuator design by using two DC motors with encapsulated gears and encoders.
- Additionally, the robot is equipped with:
  - A 32-bit microcontroller where the user control programs are executed after being downloaded into RAM via a serial line (RS-232).
  - Six infrared range sensors for the detection of obstacles.
  - A digital color camera for on-board image processing.
  - Two servos for panning the camera and for activating the ball kicking mechanism.
  - A wireless communication module to send messages to both other soccer players and a PC host system.

A.2 Underwater Robots

GARBI and *URIS* are two real underwater vehicles which can be simulated in $NEMO_{CAT}$ (as widely explained in appendix B, $NEMO_{CAT}$ [63] is a simulation tool that allows writing and testing new strategies for the autonomous navigation of robots in underwater-like scenarios). Both robots have been designed and built by the ViCOROB —Visió per COmputador i ROBòtica— research group of the University of Girona, Spain.

In the following, a brief description of the key components of the underwater vehicles GARBI and URIS is given (for further information about these vehicles, the reader is referred to [101, 102]).

A.2.1 The Vehicle *GARBI*

Generally speaking, *GARBI* was conceived as a low-cost AUV for exploration in water depths up to 200 meters. In order to accomplish a low-cost design, the vehicle was built using economic materials, such as fibre-glass and epoxy resins. The precise dimensions, in meters, of GARBI are $1.3 \times 0.9 \times 0.7$ (length $\times$ height $\times$ width). Furthermore, its maximum speed and weight are 1 knot and 150 kg, respectively.

Autonomy is certainly achieved in both control and energy since GARBI is equipped with an embedded i486 computer —running the hybrid architecture known as O^2CA^2 [103] (the Object Oriented Control Architecture for Autonomy)— as well as several battery packs. Despite such an autonomy, it is important to note that the vehicle does have an umbilical cable, which essentially covers the safety-related necessity of performing mission supervision tasks from a support vessel.

GARBI possesses four thrusters (see figure A.3): two for horizontal movements (X axis) and other two for vertical movements (Z axis). Due to the distribution of weight, the vehicle is passively stable in roll and pitch. Consequently, the number of *DOFs* is four, namely: surge, sway, heave, and yaw. Nevertheless, only surge, heave, and yaw can be directly controlled by the thrusters.

To finish, GARBI's sensors are mainly the following: two magnetic compasses, two pressure sensors, one water speed sensor, and one video camera.

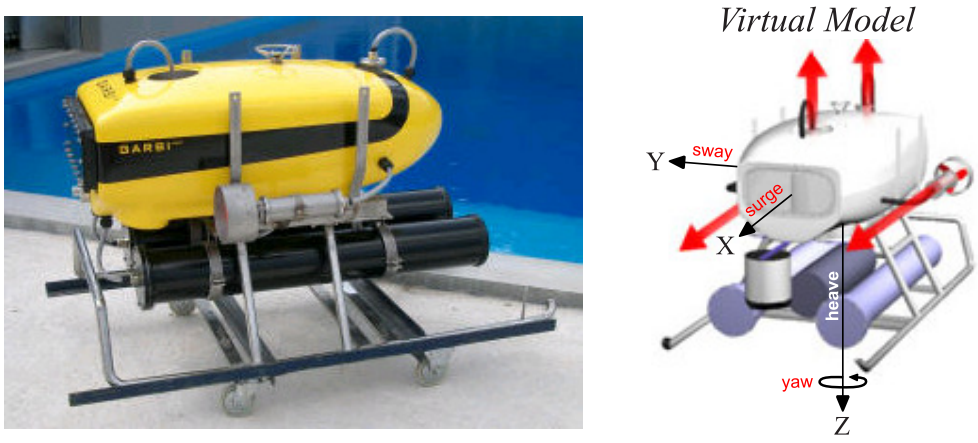


Figure A.3: The underwater vehicle GARBI.

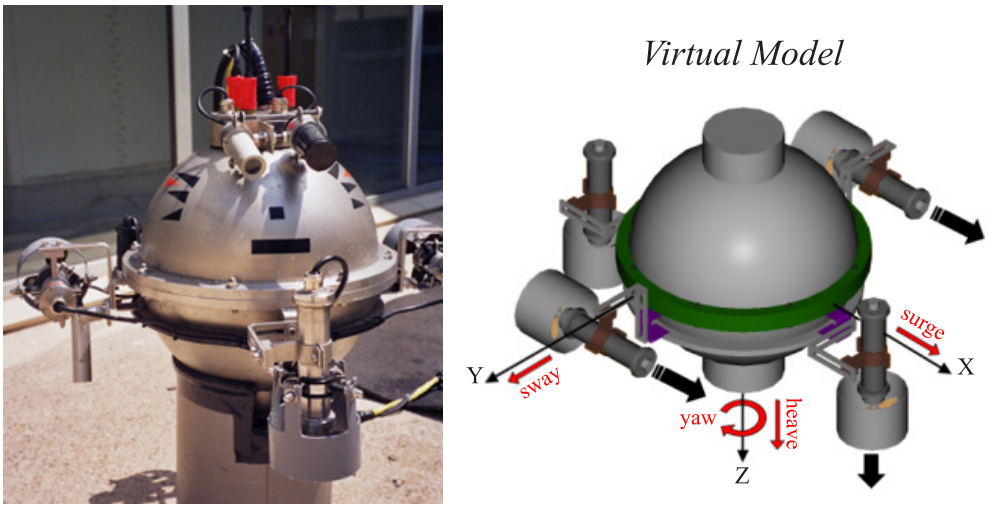


Figure A.4: The underwater vehicle URIS.

A.2.2 An Easy-to-Transport Vehicle named *URIS*

URIS, which stands for *Underwater Robotic Intelligent System*, was constructed with the intention of getting a small, light-weight, and low-cost *AUV* to be used as a research testbed in a water tank testing facility. As can be observed in figure A.4, the hull of this vehicle is composed of two stainless steel hemispheres joined with wing nuts and bolts. The sphere radius is about 17.5 cm and the weight is approximately 35 kg. Inside the hull, URIS incorporates some batteries that ensure up to an hour of autonomy. Additionally, a PC104 is in charge of the control of the vehicle based on the information provided by several on-board sensors such as, for instance, a magnetic compass, a pressure sensor, a water speed sensor, *DGPS* (*Differential Global Positioning System*), and video cameras. Propulsion is achieved by means of four thrusters placed equidistant on the exterior of the vehicle, as illustrated in figure A.4. Due to the stability of URIS in roll and pitch, there are only four *Degrees Of Freedom (DOFs)*: surge, sway, heave, and yaw. Except for sway —lateral motion—, the other *DOFs* can be controlled directly.

$NEMO_{CAT}$: A Simulator for Underwater Vehicles

In the field of robotics, a lot of valuable advantages derive from using simulation tools. The loss of detail with respect to the real world, closely bound up with simulators, is compensated by a significant reduction of the effort, risks, and monetary costs needed to carry out a series of experiments. This, nevertheless, does not mean to substitute the experimentation with prototypes as it is warned in [104], but simply to complement it.

A 3D object-oriented simulator briefly named $NEMO_{CAT}$[1] [63] has been developed in order to validate and tune reactive and hybrid control architectures for *AUVs*. Notice that both control paradigms are suitable to deal with dynamic and unstructured environments such as the submarine.

The design of this simulation tool (see figure B.1) has required the use of a software engineering methodology, the *Rational Unified Process (RUP)* [105] to be exact, which has been applied together with the *Unified Modelling Language (UML)* [106]. The latter is a general-purpose visual modeling language that permits users to specify, visualize, construct, and document the artifacts of any complex software system such as $NEMO_{CAT}$. As for the implementation of the simulator, it has been based on the C++ programming language and the OpenGL graphics library.

Next, a description of the main features of the virtual underwater environment through which vehicles navigate is provided. Additionally, the different kinds of *AUVs* as well as sensory equipment that can be faithfully simulated on $NEMO_{CAT}$ are also discussed. As a closing point, and by way of example, a behavior-based control architecture which has successfully been tested in the simulator is presented.

B.1 The Underwater Environment

As can be observed in figure B.2(a), the seabed is modeled by means of a grid of points whose extent and resolution can be configured for the mission at hand. Initially, all the points of such a grid are onto a plane which is parallel to the one defined by the X and Y axes. Typical elements of underwater environments such as rocks, holes, and algae can be afterwards added to the seabed. To this end, the heights, or Z coordinates, of some grid points are altered according to the position and shape —elliptical, in our case— of the elements incorporated. An adaptation of the well-known computer graphics algorithm called *random displacement of the midpoint* is employed so as to give to those seabed deformations a natural appearance $\Big($look at figure B.2(b)$\Big)$.

[1] $NEMO_{CAT}$ stands for *Navigational Environment MOdeler, Control Architecture Tester*

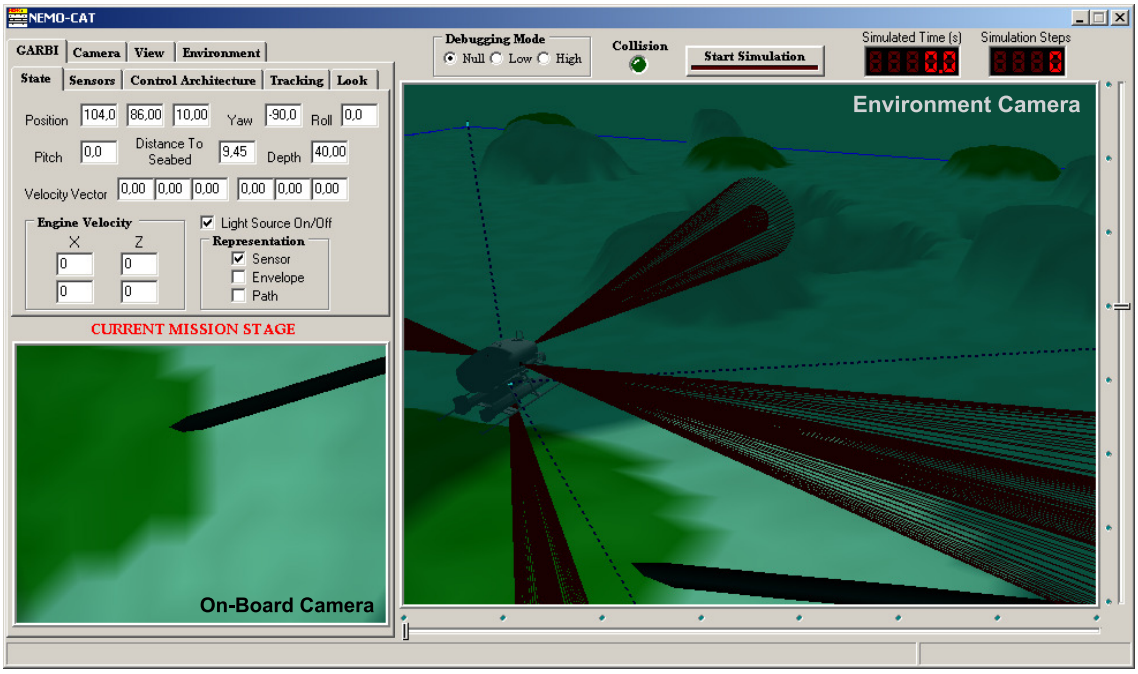
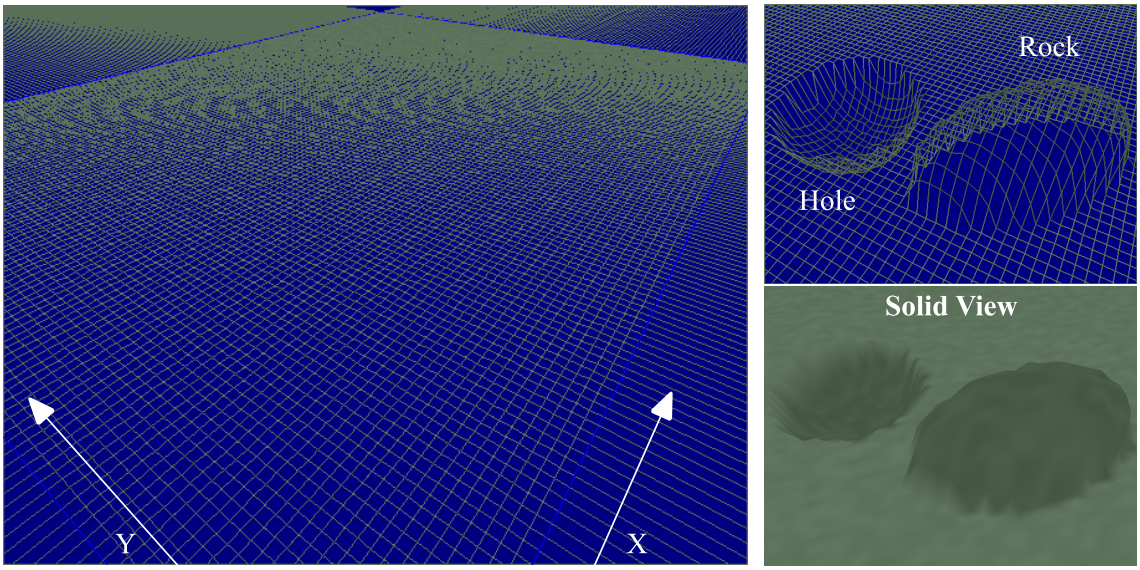


Figure B.1: A global view of $NEMO_{CAT}$. The environment where the vehicle navigates appears in the rightmost window, while the window at the left-lower corner shows the image captured by an on-board camera. Finally, at the left-upper corner, some data about the state of both the vehicle and the control architecture are displayed.



(a) (b)

Figure B.2: (a) seabed model and (b) deformations to simulate holes, rocks, ...

As it seems obvious, the simulation of distance-measurement sensors —for instance, sonars— requires the detection of the environmental obstacles that are located nearby the AUV. In $NEMO_{CAT}$, obstacles are essentially supposed to be represented by the three basic deforming elements of the seabed, namely rocks, holes, and algae[2]. To this respect, with the aim of facilitating the detection of obstacles by the virtual sensors, $NEMO_{CAT}$ approximates the actual shapes of rocks, holes, and algae, by easy-to-compute surfaces, generically referred to as *bounding* surfaces.

Besides rocks, holes, and algae, $NEMO_{CAT}$ does allow introducing user / application specific elements into the underwater environment. Just to put an example, in the current release of the simulator, cables and pipelines can also be deployed on the seabed. Moreover, common problems with these structures such as partial concealments and free span[3] can be purposely simulated.

Lastly, the reader should note that, in the electronic version of this document, there is a ≪video≫ that illustrates, step by step, all previous seabed-modeling concepts.

B.2 Autonomous Underwater Vehicles

AUVs are incorporated into the simulator by specifying their dynamic model, together with their visual appearance, and their particular sensor and actuator configurations. Since there is not a limit in relation to the maximum number of *AUVs* that can be simulated at once, multi-robot control strategies can also be studied by using $NEMO_{CAT}$.

B.2.1 The Dynamic Model

In accordance with [107, 108], the dynamics of an underwater vehicle are assumed to obey the non-linear model with six *DOFs* summarized by equation B.1, where: M_{RB} and M_A are the inertia and added-mass matrices, respectively; C_{RB} and C_A contain the Coriolis and centripetal terms associated with the two preceding matrices; v is the linear and angular velocity vector; D is the damping matrix; g represents the gravity and buoyancy forces; η denotes the position and orientation of the vehicle; and, finally, τ defines the forces and torques exerted by the vehicle's thrusters.

$$(M_{RB} + M_A)\dot{v} + (C_{RB}(v) + C_A(v))v + D(v)v + g(\eta) = \tau \tag{B.1}$$

At the moment, $NEMO_{CAT}$ bases the simulation of *AUVs* on two dynamic models, which correspond with the ones of the real underwater vehicles GARBI and URIS. These vehicles have been designed and built by the ViCOROB —Visió per COmputador i ROBòtica— research group of the University of Girona, Spain. Information about the precise method followed to estimate their dynamics can be found in [109, 110].

B.2.2 The Sensory Equipment

$NEMO_{CAT}$ supports three different kinds of sensors —sonars, compasses, and cameras—, which can be employed by the *AUVs* to suitably perform their missions. What is more, in applications where there is a need for knowing the position of the vehicle, an acoustic

[2] Algae are dangerous because they can foul the vehicle's propellers

[3] In a few words, *free span* means a portion of cable / pipeline that is not in contact with the seabed

positioning system can also be used to determine it. A résumé of the main characteristics of each of these sensory units comes next:

- As many *sonars* as desired can be attached to an AUV. The beam they generate is supposed to be conical, and its resolution, aperture, and the minimum and maximum distances that can be measured are specified as parameters.
- A magnetic *compass* is available for determining the orientation of the vehicle relative to the Earth's magnetic north pole.
- Generic pin-hole *cameras* with six *DOFs* are also available to be put on board. This fact opens up the possibility of implementing computer vision algorithms with quite different applications. As a final comment, notice that *spotlights* can be fixed onto the vehicle as well to improve the quality of the images captured by the cameras.
- The *acoustic positioning system* being adopted in $NEMO_{CAT}$ is of the so-called *Long BaseLine (LBL)* type. Concisely, an *LBL* system accurately estimates the position of an AUV relative to an array of transponders, at least three, deployed at known locations on the seabed. More exactly, such an estimation is done in two steps: first of all, the distance of the AUV to each transponder is determined by using acoustic time-of-flight measurements. Later, these distances are used to compute the position of the vehicle by simple triangulation.

At some future time, the sensory equipment of the vehicles will be significantly extended by means of pressure, speed, water and battery charge detection sensors, as well as inertial measurement units.

B.3 An Experiment

With the clear intention of showing, in practice, some of the capabilities of $NEMO_{CAT}$, a control architecture which has successfully been implemented and tested is presented. Specifically, it is intended to carry out a simple task: reach a user-defined sequence of target points while avoiding obstacles as well as getting stuck in any part of the underwater environment. In order to cope with this task, a reactive approach relying on schema theory was developed (see section 2.1.4 for an introduction to schema-based control systems). As can be observed in figure B.3, the proposed approach was composed by both three primitive behaviors and a coordination mechanism in charge of properly merging all the behavioral responses. As for the primitive behaviors, they are called *Avoid Obstacles*, *Avoiding the Past*, and *Go To*. The meaning of each behavior is outlined in the following:

- *Avoid Obstacles* allows the vehicle to circumnavigate navigational barriers such as rocks, algae, or, even, other possible cooperating vehicles. To this end, this behavior gives as output a motion vector pointing in the opposite direction to the obstacles. Moreover, the magnitude of the vector varies depending on the distance that separates the AUV from the obstacles ahead.
- *Avoiding the Past* tries to avoid the well-known local minima problem that suffers the reactive control paradigm. For such a purpose, the most recent trajectory of the AUV is understood as an obstacle that generates repulsive forces. In this way, the vehicle

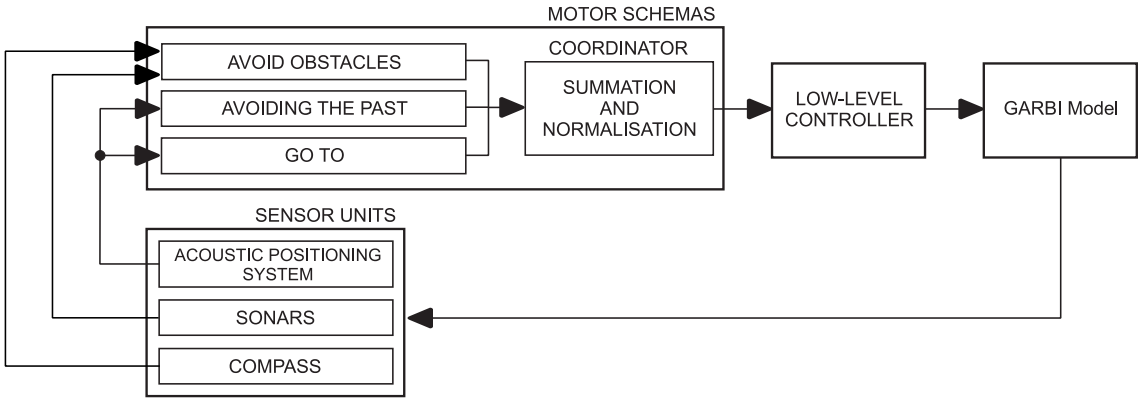


Figure B.3: Main components of the suggested control system.

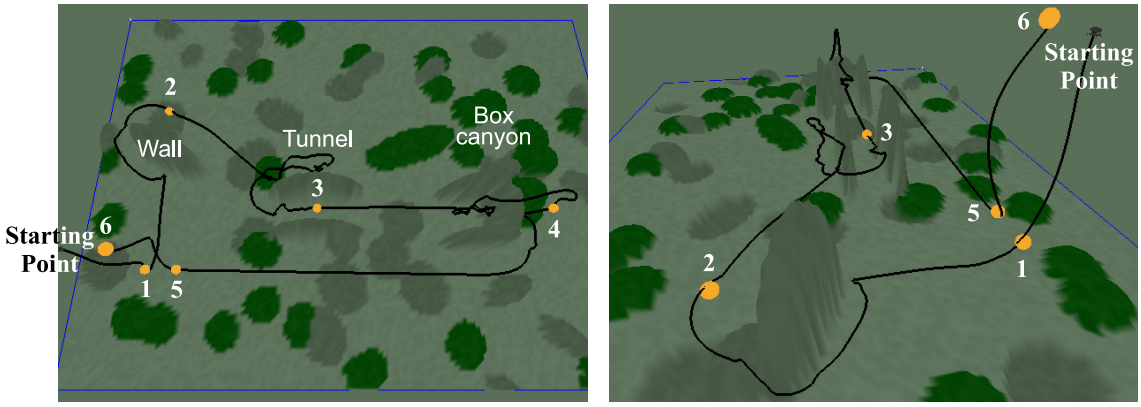


Figure B.4: Two different views showing the resultant AUV's trajectory.

is surely prevented from stalling. A more detailed description of this behavior can be found in [111], and also in section 2.1.4.

- *Go To* drives the vehicle to a certain user-specified 3-dimensional point by generating a vector, constant in magnitude, whose direction joins the current position of the AUV with the target point under consideration. In fact, this behavior is a bit more complex because it keeps a list of all those points of the environment that should be visited during the mission, following the order in which they were provided. When the vehicle gets sufficiently close to the ongoing target point, the next one in the list is chosen.

As can be anticipated from figure B.4, the intended mission consisted in the achievement of six target points spread throughout a $200 \times 150 \times 50$-metre scenario. Three main obstacles were defined to make the task of the AUV rather difficult: a wall-like rock, a long and narrow tunnel, and a box-shaped canyon. The mission was simulated using the dynamic model of the robot GARBI. As for the results, GARBI was able to fully reach the given sequence of target points in spite of the above-mentioned troublesome obstacles (see figure B.4 again). Regarding the GARBI's control system, figure B.5 depicts the activity of each behavior along the mission.

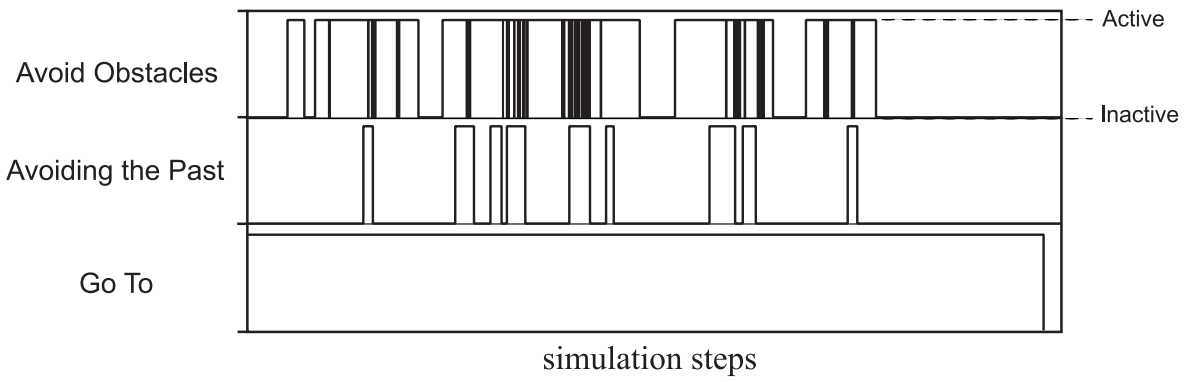


Figure B.5: Behavior activity.

Computing the Average Path Length for *Random* T^2

C.1 Randomness as a Factor Affecting Results

As described in section 3.2, under the name of *Random* T^2, there is a novel purely reactive navigation strategy which does not suffer from the *local minima* problem, in spite of being based on the classical —and non-local minima free— artificial potential fields method [8]. Furthermore, as can be easily guessed, such a strategy has a random nature in the sense that some navigational decisions are not made in a deterministic way. More exactly, randomness is introduced in the choice of the particular direction —either *left* or *right*— that the robot should take to follow / circumnavigate the contour of an obstacle that has just been detected and is impeding the progress towards the target.

From above, it seems clear that different paths might be obtained by running *Random* T^2 several times under the same experimental conditions (for us, two experiments are considered to be executed under identical conditions if they involve the same navigation environment, as well as the same location of the starting point and the same location of the target). Next, a stochastic analysis on the average length of the paths generated by the strategy *Random* T^2 is presented in order to fairly compare its results with the ones provided by other approaches.

C.2 Stochastic Analysis of the Average Path Length

Figures C.1 and C.2 show the average path length (APL) of the strategy *Random* T^2 in the seven missions that were included in the comparative study of section 3.2.4.2. By observing these figures, one can plainly deduce the general method used to determine the corresponding APL for each of the missions under consideration. This method essentially consisted of the following steps (the description assumes the APL analysis of a non-specific mission i): as a first step, (1) all paths that *Random* T^2 could generate in mission i were identified; then, (2) both the length and the probability of occurring of the preceding paths were calculated; and, to finish, (3) a sum was performed over the weighted values resulting from multiplying the probability of each path by its length.

On the practical application of the stated method, it is important to highlight that carrying out steps 1 and 2 did entail finding the so-called *random decision* points. Generally speaking, this concept is directly related to the event of detecting a new obstacle. More specifically, a random decision point represents the necessity of randomly selecting a direction to move along the contour of a new detected obstacle. With this in mind, in step 1, the search of paths was solved by examining, at each random decision point, the two possible contour following directions —namely, *left* and *right*— that *Random* T^2 could decide to take. On the other hand,

with respect to step 2, the probability of a path was given by the expression $\left(\frac{1}{2}\right)^n$, where n means the total number of random decision points along the path.

Although it has not been mentioned up until now, among the missions appearing in figures C.1 and C.2, mission 3 should be actually considered as a special case. Basically, this is because such a mission did require a way of calculating the APL that significantly differed from the one previously explained. Moreover, the underlying reason for demanding a different APL calculation was found in the infinite number of possible *Random* T^2-based paths to be exceptionally managed in that mission. In summary, an alternative method for computing the APL had to be used to cope with the infinite set of potential solutions of the strategy *Random* T^2 in mission 3. Regarding this alternative method, it consisted of three sequential steps, which were as follows: first of all, the path of the robot was both forked and divided on the basis of the random decision points $\Big($by doing so, ten path segments were obtained, labeled as $S_{A..J}$ —look at figure C.1(c)$\Big)$; afterwards, the length of each path segment S_k was multiplied by the number of times that the strategy *Random* T^2 was expected, on average, to pass over S_k; and, lastly, the resultant products were added up. Equations C.1 and C.2 formalize the outcome that was accomplished after completing the suggested steps $\Big($observe that, in these equations, the term L_{S_k} denotes the length of the path segment $S_k\Big)$.

$$\begin{aligned} APL &= L_{S_J} + \tfrac{1}{2}\left(L_{S_A} + L_{S_B} + L_{S_D} + c_r\right) + \tfrac{1}{2}\left(L_{S_C} + L_{S_D} + c_r\right) \\ &= \left(\tfrac{5}{2}L_{S_A} + \tfrac{3}{2}L_{S_B} + \tfrac{5}{2}L_{S_C} + 4L_{S_D} + 3L_{S_E} + L_{S_F} + 2L_{S_G} + L_{S_H} + 3L_{S_I} + 3L_{S_J}\right)/3 \end{aligned} \tag{C.1}$$

$$\begin{aligned} c_r &= \tfrac{1}{2}L_{S_I} + \tfrac{1}{2}\left(L_{S_E} + L_{S_G} + \tfrac{1}{2}\left(L_{S_F} + L_{S_E} + L_{S_I}\right) + \tfrac{1}{2}\left(L_{S_H} + L_{S_A} + L_{S_C} + L_{S_D} + c_r\right)\right) \\ &= \textstyle\sum_{n=0}^{\infty} L_{S_I}\left(\frac{1}{2}\right)^{2n+1} + \sum_{n=0}^{\infty}\left(L_{S_E} + L_{S_G}\right)\left(\frac{1}{2}\right)^{2n+1} + \\ &\quad \textstyle\sum_{n=1}^{\infty}\left(L_{S_F} + L_{S_E} + L_{S_I}\right)\left(\frac{1}{2}\right)^{2n} + \sum_{n=1}^{\infty}\left(L_{S_H} + L_{S_A} + L_{S_C} + L_{S_D}\right)\left(\frac{1}{2}\right)^{2n} \\ &= \left(L_{S_A} + L_{S_C} + L_{S_D} + 3L_{S_E} + L_{S_F} + 2L_{S_G} + L_{S_H} + 3L_{S_I}\right)/3 \end{aligned} \tag{C.2}$$

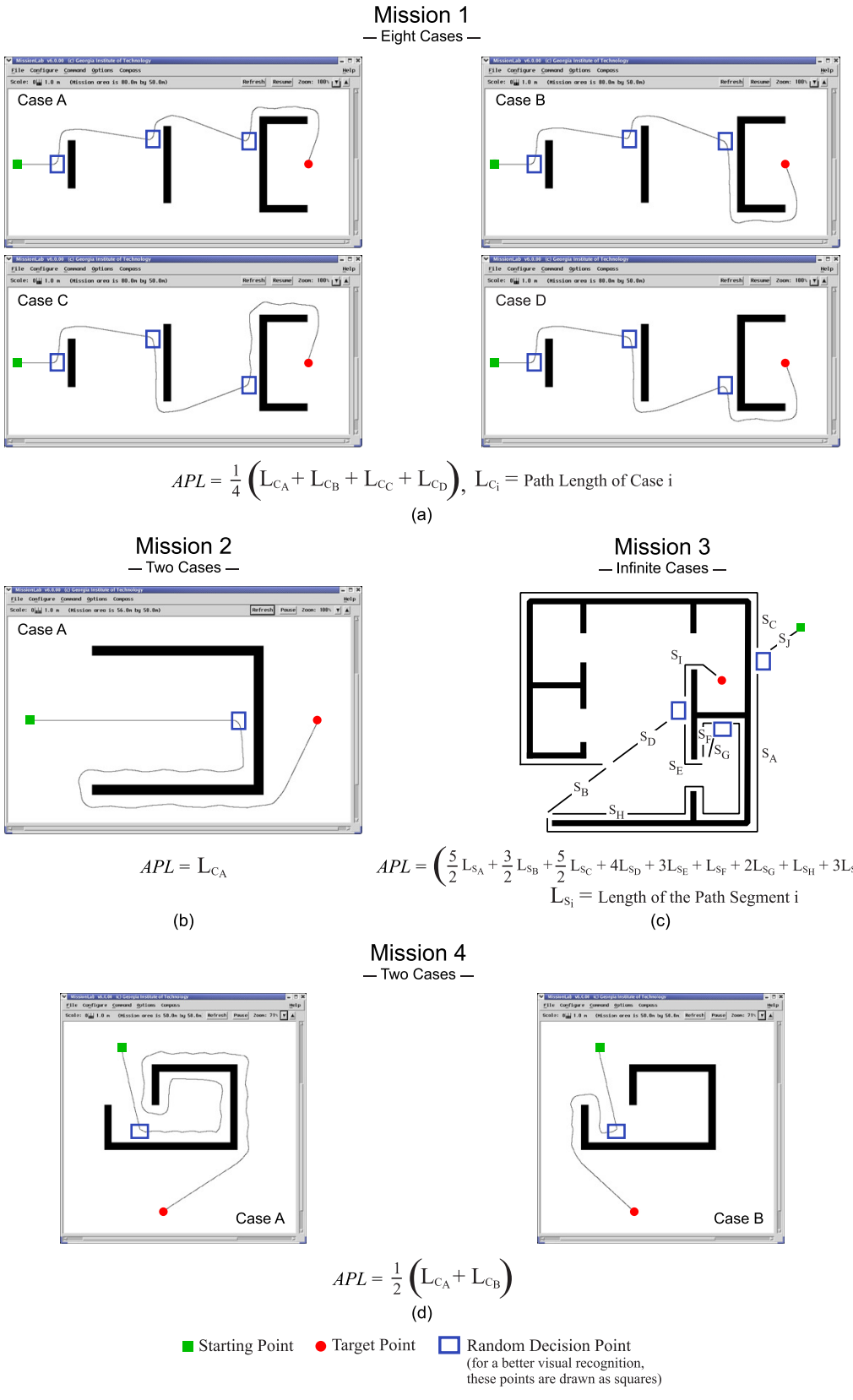


Figure C.1: Average path length (APL) of *Random* T^2 in missions 1 to 4. Observe that only half of the possible paths / cases is depicted for those missions —1 and 2, to be precise— whose environment is symmetric with regard to the line connecting the starting and the target points.

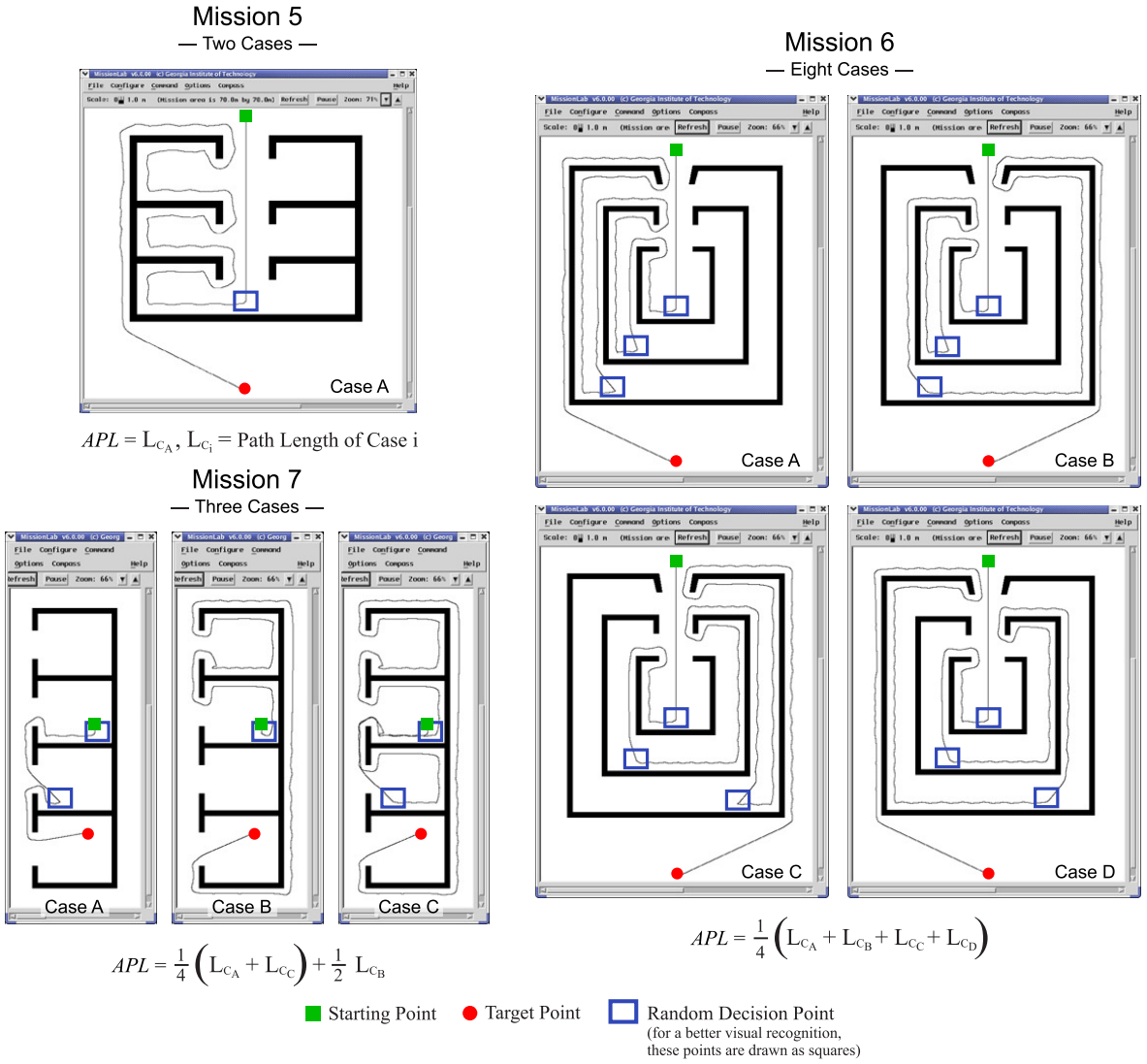


Figure C.2: Average path length (APL) of *Random* T^2 in missions 5 to 7. Observe again that only half of the possible paths / cases is depicted for those missions —5 and 6, to be precise— whose environment is symmetric with regard to the line connecting the starting and the target points.

Bibliography

[1] R. Arkin, *Behavior-based robotics*. MIT Press, 1998.

[2] P. Valavanis, D. Gracanin, M. Matijasevic, R. Kolluru, and G. Demetriou, "Control architectures for autonomous underwater vehicles," *IEEE Control Systems Magazine*, no. 17, pp. 48–64, 1997.

[3] A. Medeiros, "A survey of control architectures for autonomous mobile robots," *Journal of the Brazilian Computer Society*, vol. 4, no. 3, 1998.

[4] P. Ridao, J. Batlle, J. Amat, and G. Roberts, "Recent trends in control architectures for autonomous underwater vehicles," *Intl. Journal of Systems Science*, vol. 30, no. 9, pp. 1033–1056, 1999.

[5] E. Coste-Manière and R. Simmons, "Architecture, the backbone of robotic systems," in *Proceedings of the Intl. Conference on Robotics and Automation*, 2000, pp. 67–72.

[6] D. Nakhaeinia, S. H. Tang, S. B. Mohd Noor, and O. Motlagh, "A review of control architectures for autonomous navigation of mobile robots," *International Journal of the Physical Sciences*, vol. 6, no. 2, pp. 169–174, 2011.

[7] R. R. Murphy, *An Introduction to AI Robotics*, M. Press, Ed., 2001.

[8] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Intl. Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[9] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Proceedings of the Intl. Conference on Robotics and Automation*, 1991, pp. 1398–1404.

[10] B. Krogh, "A generalised potential field approach to obstacle avoidance," in *Proceedings of the Intl. Robotics Research Conference*, 1984, pp. 1150–1156.

[11] B. Krogh and C. Thorpe, "Integrated path planning and dynamic steering control for autonomous vehicles," in *Proceedings of the Intl. Conference on Robotics and Automation*, 1986, pp. 1664–1669.

[12] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, no. 5, pp. 1179–1187, 1989.

[13] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *Proceedings of the Intl. Conference on Robotics and Automation*, 1985.

[14] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer Magazine*, pp. 46–57, 1989.

[15] J. Borenstein and Y. Koren, "The vector field histogram: Fast obstacle avoidance for mobile robots," *IEEE Journal of Robotics and Automation*, vol. 7, no. 3, pp. 278–288, 1991.

[16] I. Ulrich and J. Borenstein, "VFH+: Reliable obstacle avoidance for fast mobile robots," in *Proceedings of the Intl. Conference on Robotics and Automation*, 1998, pp. 1572–1577.

[17] I. Ulrich and J. Borenstein, "VFH*: Local obstacle avoidance with look-ahead verification," in *Proceedings of the Intl. Conference on Robotics and Automation*, 2000, pp. 2505–2511.

[18] M. Arbib, *Perceptual Structures and Distributed Motor Control.* Oxford University Press, 1981, ch. 33, pp. 1449–1480, in Handbook of Physiology – The Nervous System II: Motor Control.

[19] R. Arkin, "Motor schema-based mobile robot navigation," *Intl. Journal of Robotics Research*, vol. 8, no. 4, pp. 92–112, 1989.

[20] T. Balch and R. Arkin, "Avoiding the past: A simple but effective strategy for reactive navigation," in *Proceedings of the Intl. Conference on Robotics and Automation*, 1993, pp. 678–685.

[21] A. Scalzo, A. Sgorbissa, and R. Zaccaria, "μNAV: a minimalist approach to navigation," in *Proceedings of the Intl. Conference on Robotics and Automation*, 2003, pp. 2018–2023.

[22] A. Sgorbissa, "Toward a multi-ethnic community of humans, mobile robots, and intelligent devices," Ph.D. dissertation, Università di Genova, 2000.

[23] A. Sgorbissa and R. Zaccaria, "The micronavigation algorithm: a minimalist approach to navigation," in *Proceedings of the Symposium on Intelligent Autonomous Vehicles*, 2004.

[24] C. Connolly, B. Burns, and R. Weiss, "Path planning using laplace's equation," in *Proceedings of the Intl. Conference on Robotics and Automation*, 1990, pp. 2102–2106.

[25] E. Rimon and D. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 501–518, 1992.

[26] M. Pearce, R. Arkin, and A. Ram, "The learning of reactive control parameters through genetic algorithms," in *Proceedings of the Intl. Conference on Intelligent Robots and Systems*, 1992, pp. 130–137.

[27] A. Ram, R. Arkin, G. Boone, and M. Pearce, "Using genetic algorithms to learn reactive control parameters for autonomous robotic navigation," *Adaptive Behavior*, vol. 2, no. 3, pp. 277–304, 1994.

[28] R. Clark, R. Arkin, and A. Ram, "Learning momentum: on-line performance enhancement for reactive systems," in *Proceedings of the Intl. Conference on Robotics and Automation*, 1992, pp. 111–116.

[29] J. Lee and R. Arkin, "Learning momentum: integration and experimentation," in *Proceedings of the Intl. Conference on Robotics and Automation*, 2001, pp. 1975–1980.

[30] J. Lee and R. Arkin, "Adaptive multi-robot behavior via learning momentum," in *Proceedings of the Intl. Conference on Intelligent Robots and Systems*, 2003, pp. 2029–2036.

[31] A. Ram, R. Arkin, K. Moorman, and R. Clark, "Case-based reactive navigation: A method for on-line selection and adaptation of reactive robotic control parameters," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 27, no. 30, pp. 376–394, 1997.

[32] M. Likhachev and R. Arkin, "Spatio-temporal case-based reasoning for behavioral selection," in *Proceedings of the Intl. Conference on Robotics and Automation*, 2001, pp. 1627–1634.

[33] M. Likhachev, M. Kaess, and R. Arkin, "Learning behavioral parameterization using spatio-temporal case-based reasoning," in *Proceedings of the Intl. Conference on Robotics and Automation*, 2002, pp. 1282–1289.

[34] J. Lee, M. Likhachev, and R. Arkin, "Selection of behavioral parameters: Integration of discontinuous switching via case-based reasoning with continuous adaptation via learning momentum," in *Proceedings of the Intl. Conference on Robotics and Automation*, 2002, pp. 1275–1281.

[35] Z. Kira and R. Arkin, "Forgetting bad behavior: Memory management for case-based navigation," in *Proceedings of the Intl. Conference on Intelligent Robots and Systems*, 2004, pp. 3145–3152.

[36] M. Krishna and P. Kalra, "Solving the local minima problem for a mobile robot by classification of spatio-temporal sensory sequences," *Journal of Robotic Systems*, vol. 17, no. 10, pp. 549–564, 2000.

[37] H. Maaref and C. Barret, "Sensor-based navigation of a mobile robot in an indoor environment," *Robotics and Autonomous Systems*, no. 38, pp. 1–18, 2002.

[38] A. Zhu and S. Yang, "A fuzzy logic approach to reactive navigation of behavior-based mobile robots," in *Proceedings of the Intl. Conference on Robotics and Automation*, 2004, pp. 5045–5050.

[39] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *Proceedings of the Intl. Conference on Robotics and Automation*, 1999, pp. 341–346.

[40] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," CS Department, University of Bonn, Tech. Rep. IAI-TR-95-13, 1995.

[41] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics and Automation Magazine*, vol. 1, no. 4, pp. 23–33, 1997.

[42] C. Stachniss and W. Burgard, "An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments," in *Proceedings of the Intl. Conference on Intelligent Robots and Systems*, 2002, pp. 508–513.

[43] P. Ogren and N. Leonard, "A tractable convergent dynamic window approach to obstacle avoidance," in *Proceedings of the Intl. Conference on Intelligent Robots and Systems*, 2002, pp. 595–600.

[44] P. Ogren and N. Leonard, "A convergent dynamic window approach to obstacle avoidance," *IEEE Transactions on Robotics and Automation*, vol. 21, no. 2, pp. 188–195, 2005.

[45] J. Fernández, R. Sanz, J. Benayas, and A. Diéguez, "Improving collision avoidance for mobile robots in partially known environments: The beam curvature method," *Robotics and Autonomous Systems*, no. 46, pp. 205–219, 2004.

[46] N. Ko and R. Simmons, "The lane-curvature method for local obstacle avoidance," in *Proceedings of the Intl. Conference on Intelligent Robots and Systems*, 1998, pp. 1615–1621.

[47] R. Simmons, "The curvature-velocity method for local obstacle avoidance," in *Proceedings of the Intl. Conference on Robotics and Automation*, 1996, pp. 2275–2282.

[48] J. Mínguez and L. Montano, "Nearness diagram (ND) navigation: Collision avoidance in troublesome scenarios," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 1, pp. 45–59, 2004.

[49] J. Mínguez, J. Osuna, and L. Montano, "A divide and conquer strategy based on situations to achieve reactive collision avoidance in troublesome scenarios," in *Proceedings of the Intl. Conference on Robotics and Automation*, 2004, pp. 3855–3862.

[50] J. Mínguez, "The obstacle-restriction method (ORM) for robot obstacle avoidance in difficult environments," in *Proceedings of the Intl. Conference on Intelligent Robots and Systems*, 2005, pp. 3706–3712.

[51] J. Durham and F. Bullo, "Smooth nearness-diagram navigation," in *Proceedings of the Intl. Conference on Intelligent Robots and Systems*, 2008, pp. 690–695.

[52] M. Mujahad, D. Fischer, B. Mertsching, and H. Jaddu, "Closest gap based (cg) reactive obstacle avoidance navigation for highly cluttered environments," in *Proceedings of the Intl. Conference on Intelligent Robots and Systems*, 2010, pp. 1805–1812.

[53] V. Lumelsky and A. Stepanov, "Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape," *Algorithmica*, no. 2, pp. 403–430, 1987.

[54] V. Lumelsky, "A comparative study on the path length performance of maze-searching and robot motion planning algorithms," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 1, pp. 57–66, 1991.

[55] V. Lumelsky and T. Skewis, "Incorporating range sensing in the robot navigation function," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, no. 5, pp. 1058–1069, 1990.

[56] I. Kamon and E. Rivlin, "Sensory-based motion planning with global proofs," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 6, pp. 814–822, 1997.

[57] I. Kamon, E. Rimon, and E. Rivlin, "TangentBug: A range-sensor based navigation algorithm," *Intl. Journal of Robotics Research*, vol. 9, no. 17, pp. 934–953, 1998.

[58] Y. Gabriely and E. Rimon, "CBUG: A quadratically competitive mobile robot navigation algorithm," in *Proceedings of the Intl. Conference on Robotics and Automation*, 2005, pp. 2026–2031.

[59] I. Kamon, E. Rimon, and E. Rivlin, "Range-sensor based navigation in three dimensions," in *Proceedings of the Intl. Conference on Robotics and Automation*, 1999, pp. 163–169.

[60] I. Kamon, E. Rimon, and E. Rivlin, "Range-sensor-based navigation in three-dimensional polyhedral environments," *Intl. Journal of Robotics Research*, vol. 20, no. 1, pp. 6–25, 2001.

[61] Y.-C. Lin, C.-C. Chou, and F.-L. Lian, "Indoor robot navigation based on DWA*: Velocity space approach with region analysis," in *Proceedings of the ICROS-SICE Intl. Joint Conference*, 2009, pp. 700–705.

[62] M. Khatib, "Sensor-based motion control for mobile robots," Ph.D. dissertation, LAAS-CNRS, 1996.

[63] J. Antich and A. Ortiz, "An underwater simulation environment for testing autonomous robot control architectures," in *proceedings of the IFAC conference on Control Applications in Marine Systems*, July 2004, Ancona (Italy), pp. 509–514, ISSN 1474-6670.

[64] D. Mackenzie, R. Arkin, and J. Cameron, "Multiagent mission specification and execution," *Autonomous Robots*, no. 4, pp. 29–52, 1997.

[65] D. Mackenzie and R. Arkin, "Evaluating the usability of robot programming toolsets," *Intl. Journal of Robotics Research*, vol. 4, no. 17, pp. 381–401, 1998.

[66] Y. Endo, D. Mackenzie, and R. Arkin, "Usability evaluation of high-level user assistance for robot mission specification," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 34, no. 2, pp. 168–180, 2004.

[67] R. Arkin and T. Balch, "AuRA: Principles and practice in review," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9, no. 2-3, pp. 175–188, 1997.

[68] J. Osca, "Implementación y test de nuevas estrategias de navegación para vehículos autónomos en MissionLab," 2005, graduation project, University of the Balearic Islands.

[69] A. Ranganathan and S. Koenig, "A reactive robot architecture with planning on demand," in *Proceedings of the Intl. Conference on Intelligent Robots and Systems*, 2003, pp. 1462–1468.

[70] C. Chih-Chung, L. Feng-Li, and W. Chieh-Chih, "Characterizing indoor environment for robot navigation using velocity space approach with region analysis and look-ahead verification," *IEEE Transactions on Instrumentation and Measurement*, vol. 60, no. 2, pp. 442–451, 2010.

[71] J. Ng and T. Bräunl, "Performance comparison of Bug navigation algorithms," *Journal of Intelligent and Robotic Systems*, vol. 50, no. 1, pp. 73–84, 2007.

[72] W. Massey, *Algebraic Topology: An Introduction*. Harcourt, Brace & World, 1967.

[73] R. Zhou and E. Hansen, "Multiple sequence alignment using A*," in *Proc. of the National Conference on Artificial Intelligence*, 2002.

[74] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *Proceedings of Advances in Neural Information Processing Systems*. MIT Press, 2003.

[75] D. Ferguson and A. Stentz, "Anytime RRTs," in *Proceedings of the Intl. Conference on Intelligent Robots and Systems*, 2006, pp. 5369–5375.

[76] P. Hart, N. Nilsson, and B. Rafael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE T-SSC*, 1968.

[77] J. Pearl, *Heuristics*. Addison-Wesley, 1984.

[78] S. Bespamyatnikh, "Computing homotopic shortest paths in the plane," *Journal of Algorithms*, vol. 49, no. 2, pp. 284–303, 2003.

[79] D. Ferguson, M. Likhachev, and A. Stentz, "A guide to heuristic-based path planning," in *Proc. of the workshop on Planning under Uncertainty for Autonomous Systems at ICAPS*, 2005.

[80] J. Barraquand, B. Langlois, and J. Latombe, "Numerical potential field techniques for robot path planning," *IEEE Transactions on Man and Cybernetics*, vol. 22, no. 2, pp. 224–241, 1992.

[81] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Intl. Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[82] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[83] J. Antich and A. Ortiz, "T^2: An approach to robotic navigation in unknown and dynamic environments," Department of Mathematics and Computer Science, University of the Balearic Islands, Tech. Rep. A-3, 2004.

[84] J. Antich, "Reactive robotics: A paradigm not limited to simple tasks," May 2006, Research Report, Department of Mathematics and Computer Science, University of the Balearic Islands.

[85] J. Antich and A. Ortiz, "A dynamic window approach to navigate in complex scenarios using low-cost sensors for obstacle detection," Department of Mathematics and Computer Science, University of the Balearic Islands, Tech. Rep. A-4, 2007.

[86] J. Antich and A. Ortiz, "Bug2+: Details and formal proofs," Department of Mathematics and Computer Science, University of the Balearic Islands, Tech. Rep. A-1, 2009.

[87] J. Antich and A. Ortiz, "Extending the potential fields approach to avoid trapping situations," in *proceedings of the IEEE/RSJ international conference on Intelligent Robots and Systems*, August 2005, Edmonton (Canada), pp. 1379–1384, ISBN 0-7803-8913-1.

[88] J. Antich and A. Ortiz, "Bug-based T^2: A new globally convergent potential field approach to obstacle avoidance," in *proceedings of the IEEE/RSJ international conference on Intelligent Robots and Systems*, October 2006, Beijing (China), pp. 430–435, ISBN 1-4244-0259-X.

[89] J. Antich and A. Ortiz, "A convergent dynamic window approach with minimal computational requirements," in *proceedings of the 10th international conference on Intelligent Autonomous Systems*, July 2008, Baden Baden (Germany), pp. 183–192, ISBN 978-1-58603-887-8.

[90] J. Antich, A. Ortiz, and J. Mínguez, "ABUG: A fast bug-derivative anytime path planner with provable suboptimality bounds," in *proceedings of the 14th International Conference on Advanced Robotics*, June 2009, Munich (Germany), pp. 1–8, ISBN 978-1-4244-4855-5.

[91] J. Antich, A. Ortiz, and J. Mínguez, "A bug-inspired algorithm for efficient anytime path planning," in *proceedings of the IEEE/RSJ international conference on Intelligent Robots and Systems*, October 2009, St. Louis (USA), pp. 5407–5413, ISBN 978-1-4244-3804-4.

[92] J. Antich and A. Ortiz, "A rapid anytime path planner with incorporated range sensing to improve control on solution quality," in *proceedings of the 11th international conference on Intelligent Autonomous Systems*, August 2010, Ottawa (Canada), pp. 207–216, ISBN 978-1-60750-612-6.

[93] J. Antich and A. Ortiz, "Development of the control architecture of a vision-guided underwater cable tracker," *Intl. Journal of Intelligent Systems*, vol. 20, no. 5, pp. 477–498, 2005, ISSN 0884-8173.

[94] J. Antich, A. Ortiz, and G. Oliver, "A PFM-based control architecture for a visually guided underwater cable tracker to achieve navigation in troublesome scenarios," *Journal of Maritime Research*, vol. 2, no. 1, pp. 33–50, 2005, ISSN 1697-4840.

[95] J. Antich and A. Ortiz, "Reactive navigation in troublesome environments: T^2 strategies," *Instrumentation Viewpoint*, no. 6, pp. 51–52, 2007, ISSN 1886-4864.

[96] J. Antich, A. Ortiz, and G. Oliver, *Reactive Control of a Visually Guided Underwater Cable Tracker*. In book *Robotics and Automation in the Maritime Industries* published by *Instituto de Automática Industrial* (CSIC), 2006, ch. 6, pp. 111–132, IBSN 84-611-3915-1.

[97] J. Antich and A. Ortiz, *Traversability and Tenacity: Two New Concepts Improving the Navigation Capabilities of Reactive Control Systems*. In book *Robotics and Automation in the Maritime Industries* published by *Instituto de Automática Industrial* (CSIC), 2006, ch. 7, pp. 133–154, ISBN 84-611-3915-1.

[98] E. Hernández, M. Carreras, J. Antich, P. Ridao, and A. Ortiz, "A topologically guided path planner for an AUV using homotopy classes," in *proceedings of the IEEE International Conference on Robotics and Automation*, May 2011, Shanghai (China), pp. 2337–2343, ISBN 978-1-61284-385-8.

[99] E. Hernández, M. Carreras, P. Ridao, J. Antich, and A. Ortiz, "A search-based path planning algorithm with topological constraints. Application to an AUV," in *proceedings of the 18th IEEE/OES IFAC World Congress*, August 2011, Milano (Italy), pp. 13 654–13 659, ISBN 978-3-902661-93-7.

[100] T. Bräunl, *Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems*. Springer-Verlag, 2003.

[101] J. Amat, A. Monferrer, J. Batlle, and X. Cufi, "GARBI: A low-cost underwater vehicle," *Microprocessors and Microsystems*, vol. 23, no. 2, pp. 61–67, 1999.

[102] J. Batlle, P. Ridao, R. Garcia, M. Carreras, X. Cufi, A. El-Fakdi, D. Ribas, T. Nicosevici, E. Batlle, G. Oliver, A. Ortiz, and J. Antich, *URIS: Underwater Robotic Intelligent System*. In book *Automation for the Maritime Industries* published by *Instituto de Automática Industrial* (CSIC), 2004, ch. 11, pp. 177–203, IBSN 84-609-3315-6.

[103] P. Ridao, J. Batlle, and M. Carreras, "O^2CA^2, a new object oriented control architecture for autonomy: the reactive layer," *Control Engineering Practice*, vol. 10, no. 8, pp. 857–873, 2002.

[104] R. Arkin, *Behavior-based robotics*. MIT Press, 1998.

[105] P. Kruchten, *The Rational Unified Process: An Introduction*. Addison-Wesley Professional, 2003.

[106] G. Booch, I. Jacobson, and J. Rumbaugh, *The Unified Modeling Language User Guide*. Addison-Wesley Professional, 2005.

[107] T. Fossen, *Guidance and Control of Ocean Vehicles*. John Wiley & Sons, 1994.

[108] T. Fossen, *Underwater Robotic Vehicles: Design and Control*. TSI Press, 1995.

[109] P. Ridao, J. Batlle, and M. Carreras, "Dynamics model of an underwater robotic vehicle," University of Girona, Tech. Rep. IIiA 01-05-RR, 2001.

[110] P. Ridao, A. Tiano, A. El-Fakdi, M. Carreras, and A. Zirilli, "On the identification of non-linear models of unmanned underwater vehicles," *Control Engineering Practice*, vol. 12, pp. 1483–1499, 2004.

[111] T. Balch and R. Arkin, "Avoiding the past: A simple but effective strategy for reactive navigation," in *proceedings of the IEEE International Conference on Robotics and Automation*, 1993, pp. 678–685.

Glossary

$NEMO_{CAT}$ Navigational Environment Modeller, Control Architecture Tester. 62, 191, 193, 195, 196

T^2 Traversability and Tenacity. 7, 8, 39, 40, 42, 44–47, 49, 65–74, 79–88, 129, 131, 132, 137, 139, 142, 145, 147, 152, 153, 155, 156, 184, 185, 188, 199, 200

μNAV Micronavigation. 19, 21, 67, 70, 72, 73

ABUG Anytime BUG-based path planner. 8, 160, 161, 163, 164, 166–173, 186, 188

APL Average Path Length. 70, 199, 200

ARA* Anytime Repairing A*. 169–173, 179, 180, 182, 186

ARRT Anytime Rapidly-exploring Random Tree. 169, 171, 179, 180, 182, 186

AUV Autonomous Underwater Vehicle. 3, 191–193, 195–197

BCM Beam-Curvature Method. 26

CBR Case-Based Reasoning. 24, 68–71

CFD Contour Following Direction. 55

CG Closest Gap. 30

cstr constraint. 117

CVM Curvature-Velocity Method. 26

DGPS Differential Global Positioning System. 192

DOF Degree Of Freedom. 191, 192, 195, 196

DWA Dynamic Window Approach. 25, 26, 40, 41, 73–79, 81, 85, 86, 185

GA genetic algorithm. 22

GPFM Generalized Potential Fields Method. 14

HPF Harmonic Potential Field. 21

LBL Long BaseLine. 196

LCM Lane-Curvature Method. 26

LM Learning Momentum. 22, 23, 67, 69–71

LMF Local Minima-Free. 77

MS motor schema. 17–19

ND Nearness Diagram method. 27–29, 43, 44

NF1 Wave Front. 169–173

oml open m-line. 104, 106–108, 110, 111, 115, 117, 119, 121–123, 125, 126

ORM Obstacle-Restriction Method. 27–29

PFM Potential Fields Method. 11, 13, 21, 40, 54, 62–64, 73, 81, 185

ROV Remotely Operated Vehicle. 3

RRT Rapidly-exploring Random Tree. 169–173, 180

RUP Rational Unified Process. 193

SND Smooth Nearness-Diagram method. 29, 30

UML Unified Modelling Language. 193

UUV Untethered Underwater Vehicle. 3

vABUG ABUG with incorporated vision. 8, 160, 161, 171, 174, 177–180, 182, 186, 188

VFF Virtual Force Field. 14–16

VFH Vector Field Histogram. 16