Robot Shape, Kinematics, and Dynamics in Sensor-Based Motion Planning

Javier Mínguez Zafra

Ph.D. Dissertation

Main advisor D. Luis Montano Gella

Departamento de Informática e Ingeniería de Sistemas
Centro Politécnico Superior
Universidad de Zaragoza, España

July 2002

Acknowledgments

I always suspected that this Section would be the last of my thesis, but I never thought that it could be so difficult. I owe gratitude to all the people that helped me to turn a dream into reality. As I am sure that I am going to forget somebody, for you then, thanks.

The first person that comes to mind is my advisor Luis Montano, who has been suffering me for the last years. With his patience, knowledge, vision, willingness to share, now I am at the end of this long way. I am very grateful for allowing me do what I love, and for believe in me and my research from the very beginning, when nobody else did. Thanks Luis.

I am very fortunate to have had the opportunity to work and interact in the Zaragoza research team. Their constructive criticism showed me how to overcome the difficulties of a thesis. Thanks to all the members of the *Robotics, Vision, and Real-Time Group*. In particular, I owe gratitude to Carlos Sagües who guided me in the first steps of the research. I want to thank my colleges J. R, Luis, Dieguito, Jorge, Eduardo, DieRodri, Merse, and Laurita. For being always there, from the beginning to the end.

Raja Chatila gave me the opportunity to work in his research group at LAAS-CNRS, France. Rachid Alami and Nicola Simeon, my advisors there, followed my work with great interest, and gave me the best research advice with a high human quality. Sara Fleury, never too far from the robots, had the patience to work with me during hundred of hours. I am still learning from all you, thanks. I also had the chance to work with many people in narrow friendship as Juanito, Ben, Patrick, Bernard and Cecile, and Nestor.

José Santos-Victor luckily accepted me in his research group, the VisLab. Thus, I traveled to Lisbon, Portugal to work with a marvelous research team. Thanks José for your human quality, for helping me in the worst moments, for helping me to land in the best moments, for your advise, and for taking me out of the lab when it was necessary. I worked and traveled in Lisbon with wonderful people that always were ready to work, to help, to go out .... Thanks Raquelinha, Sjoerd and Sonia, Niall, Eval, Gaspar, Alex, Etienne, Fofinha!, Eduardita la morenita, Nuno and Patricia, Joao, Antonio Ostia!, and Dieguito.

I want to thanks Oussama Khatib for hosting me in my visit to the Stanford University, USA. There, many people contributed to this thesis with their energy and helpful comments, especially Oliver Brock and Jaeheung Park. I survived as I could due to my friends Oli, Laurenza, Luis and Adela, Sriram, and Costas.

Towards my parents, Juan Manuel and María Victoria, I feel the most proud son of the world. They unconditionally inculcate me the life values. I wish I learnt. I still can remember the day that I told you *"I do not want to study medicine, I want to study physics and become a researcher"*. Thanks you Papas for allowing me to walk this way. And what about Juanito and Vicky? Always fighting among us, but always loving us.

Brockita, she is always there. Always present with a smile and encouragement in her lips. Thanks Broo.

I owe my gratitude to my friends Chemita and Silvi, Pablito and Mari, Villablino and Pi. You always are faithful, taking care of me. Many friends simply never forget me and I never forget them. They made me grow with them, and even that I did not wanted, I am finally doing. Thanks to my friends of Zaragoza Manu, Buti and Susi, Bon and Paloma, Ondi, Mario, Nachete, Fernandos, Diego, Polansky, Carlos, Gato, Sara and Sergio .... Thanks to my friends of Madrid Peraita, Buho, Lion, Pezzi .... And to my inseparable neighbor Jorgito!.

Contents

Chapter 1

Introduction

There are many applications in Robotics where robots are required to be safely moved in an environment to accomplish a given task. The motion generation is an important module that, in most cases, determines the success of the complete mission. Failures in this module might also have fatal consequences upon the robot and the environment.

To move a robot among locations requires to address many robotic problems. In the basic version of this task, the environment is roughly known, and the robot is required to safely reach a given location. Firstly, a collision-free path joining the current and final locations is computed: *Motion planning* [35]. Next, the robot executes the motion as close as possible to the computed path. Sensor information is needed to have a feedback of the environment and robot state. The sensory information is used to: (1) plan local motions in order to avoid non-predictable obstacles over the initial path (*Sensor-based Motion Planning*); and (2) to integrate new information into a map of the environment in order to compute the robot current location (*Simultaneous Localization and Map Building* [18], [17], [38], [20], [57]). All these issues must be addressed within a robotic architecture to build a robust navigation system (*Supervision* [51], [15], [32]). The development of robust navigation systems able to deal with everyday environments is still an open research area in Robotics.

This dissertation addresses the *Sensor-based Motion Planning* for mobile robots. These algorithms are used to locally move a robot among successive locations of a given path, while avoiding collisions with the sensed environment. This module is crucial in the complete robot architecture: a failure usually leads to collisions, or the robot invades areas where it is susceptible to be lost. Then, it is difficult to devise a real mobile robotic application that do not rely on a robust *Sensor-based Motion Planning* module.

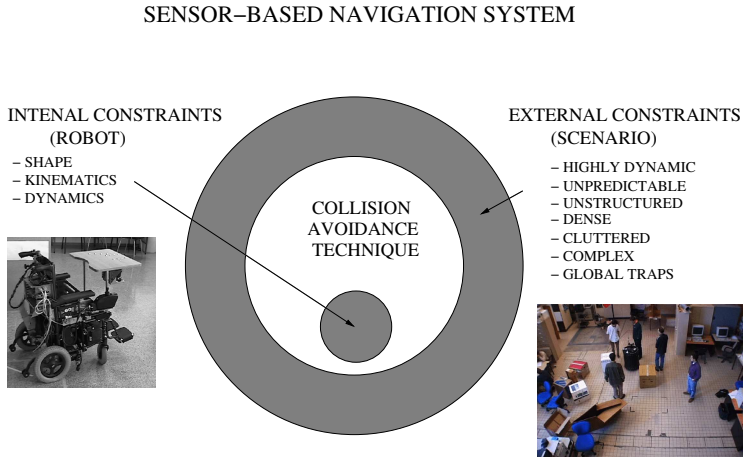


Figure 1.1: The Sensor-based navigation module.

The *Sensor-Based Motion Planning* system is governed by a *perception-action* process. *Perception* (sensory information) provides the system with a feedback of the external and internal robot state. This information is processed by the *sensor-based navigation* module to calculate an *action*, that is the execution of a motion command. Subsequently, the process is resumed.

The *sensor-based navigation* module design mainly covers at least the following three issues, see Fig. 1.1:

1. *The collision avoidance technique*: The most popular techniques of *Sensor-Based Motion Planning* are the reactive navigation methods. These methods compute at each cycle-time a collision-free motion command, to converge the current robot location towards the goal location. The result of applying these algorithms is a sequence of motions that drive the robot free of collisions towards the goal location. Examples of these methods include [30], [58], [29], [9], [50], [22], [11], [28], [56], [25], [54], [44]. The reactive navigation methods are the inner part of the *sensor-based navigation system*, see Fig. 1.1.

   Some related work solves the problem by making physical analogies. Other methods compute sets of "suitable" motion commands, to select "the best one" later on. These methods obtain good behavior when the scenario is not troublesome: the obstacle distribution is not cluttered, dense, or complex. Otherwise, these methods exhibit difficulties to successfully navigate. This issue is addressed in this thesis: robot

navigation in very dense, complex, and cluttered scenarios. Chapter 2 presents a method called *Nearness Diagram Navigation*. By using this reactive navigation method, navigation is successfully achieved in such challenge scenarios.

2. *The internal constraints*: These constraints are imposed by the vehicle specific characteristics. There are at least three types of internal constraints: (1) the robot shape, (2) the robot kinematic constraints, and (3) the robot dynamic constraints. These constraints are called internal, since they have to be considered in the design of the collision avoidance technique, see Fig. 1.1. (e.g. dealing with a car-like robot, the kinematic constraints have to be taken into account in the design of the reactive navigation method.) To address the specific vehicle shape, kinematics, and dynamics is still a subject of research in *Sensor-based Motion Planning*. However, some methods have been proposed dealing with the shape and kinematics [25], [56], [59], [60], [54], [62], [28], [22], [52], [47]; and dealing with dynamics [56], [25], [13], [22], [62], [46]. Some of them address all of these constraints [52], [16], [62].

   Mainly, related work addresses the kinematics or the dynamics by calculating sets of trajectories that comply with the kinematic or dynamic constraints. One trajectory is selected with some criteria. Subsequently, a motion command, that drives the platform over the selected trajectory, is computed and executed by the vehicle. The limitation of these approaches is that usually, the reactive navigation method is designed from scratch to address the kinematics or the dynamics. There is a lot of literature that proposes particular solutions to address the kinematic and dynamic constraints in the *Sensor-Based Motion Planning* field. But little attention has been paid to look for much wider solutions.

   The Chapter 3 presents an under-constraint solution to address the robot shape, kinematics, and dynamics in the reactive navigation layer. This solution is used to extend the *Nearness Diagram Navigation* method to address these issues. Moreover, the usage of different sensors for navigation is explored. By using this framework, navigation is successfully achieved in troublesome scenarios with non-circular robots that exhibit kinematic and dynamic constraints.

   Chapter 4 presents a solution to address the kinematic constraints based on a spatial representation called the *Ego-Kinematic space*. The robot kinematics is used in the construction of the spatial representation. In this space the robot moves as a free-flying object. Many existing reactive

navigation methods that do not address kinematics can be used in this space. As a consequence, the reactive method solution complies with the kinematic constraints. This is an ample solution to address the kinematics in the *Sensor-Based motion Planning* field.

Chapter 5 addresses the dynamic constraints with a similar idea: to introduce the dynamics into the spatial representation, the *Ego-Dynamic space*. With minor modifications, standard reactive navigation methods that do not address dynamics can be used in this space. As a consequence, the reactive method solution complies with the dynamic constraints. This is an ample solution to address the dynamic constraints into the *Sensor-Based Motion Planning* field.

3. *The external constraints*: These constraints are imposed by the type of scenario where the vehicle is moving. For most of the mobile robotic applications, the environment nature is non-predictable, unstructured, and dynamic. Moreover, the environment structure can produce trap situations or cyclic behaviors in the navigation systems that have to be avoided. These constraints are called external, since they do not directly affect the design of the collision avoidance technique. However, an external layer, that process information to improve the pure reactive behavior, is required (see Fig. 1.1). (e.g. moving the robot in a highly dynamic scenario requires a module to process the sensory information, in order to improve the reactive method behavior. Although, the reactive method might not be redesigned.) This issue has been mainly addressed in *Sensor-Based Motion Planning* breaking down the problem into sub-problems: the collision avoidance technique, and the design of the external layer that improves the collision avoidance task. Each of the modules works independently, but they interact to complete the navigation task. Examples of these methods include [11], [59], [60], [13].

   This issue has been mainly addressed by designing particular modules in particular contexts. However, the instances of the modules are not general, and no procedure is proposed to integrate the different modules. Chapter 6 presents some instances of the functionalities required to improve the method behavior in highly dynamic scenarios, and to avoid the common trap situations. Nevertheless, an architecture is presented allowing for integration of the base functionalities, and for module replacement - indispensable to reutilize the technologies proposed in this thesis for different robot and sensors.

This dissertation addresses all the issues that make up the design of the

Sensor-Based Motion Planning module: (1) the design of a sensor-based motion planner. (2) Solutions for the sensor-based motion planning with shape, kinematic, and dynamic constraints. (3) Development of the modules required to improve the performance of the sensor-based motion planner. (4) Integration of all the functionalities and the algorithms proposed in the dissertation.

Chapter 2

Nearness Diagram Navigation

2.1 Introduction

The safe motion generation is an important task in a typical indoor-outdoor mobile robotic mission. Unfortunately the robot navigation in very cluttered, dense, and complex scenarios is still a robotic challenge. These scenarios are where the robots are usually required to move. Then, the robots are limited to work in well-controlled environments.

The sensor-based motion planning algorithms are one alternative to safely drive a robot. These techniques are based on a perception-action process. The sensors collect information from the robot and the environment. The actions are the execution of safe motion commands. There are many sensor-based motion planning algorithms that have been developed in the last few years. A troublesome scenario for many sensor-based motion planners is depicted in Fig. 2.1. The robot is required to move among random distributions of any shaped obstacles like humans, doors, chairs, tables, wardrobes, and filing cabinets. This is the main subject and motivation of this Chapter: to safely move a robot in an scenario similar to the Figure.

The first contribution of this Chapter is the design of a reactive navigation method. The design has been developed using a classic design paradigm, the *situated-activity* paradigm [3]. The advantage of this paradigm is that it employs a "divide and conquer" strategy to reduce the difficulty of the main task (reactive navigation in this Chapter). Moreover, by construction, the design does not suffer from the "action coordination problem" that arises when the main task is divided into subtasks (and they have to be somehow coordinated). Reactive navigation methods are implemented following this design. The new reactive navigation methods inherit the design paradigm

Figure 2.1: Typical office environment. The snapshot was taken in a experiment performed using the *Robels* system. The *Nearness Diagram Navigation* is the sensory-motor function that is driving the robot out of the office.

advantages. As a consequence, these methods must be able to solve more complex scenarios problems than other methods do.

The *Nearness Diagram Navigation* method has been implemented following the reactive navigation method design proposed. This reactive navigation method is a geometry-based implementation. The main contribution of this implementation is that it solves highly complex navigation problems. The *Nearness Diagram Navigation* successfully achieves navigation in very complex, dense, and cluttered scenarios. These scenarios present a high degree of difficulty for many existing methods. Experimental results with a real platform are presented to validate the method in these challenge environments.

This research was previously presented in [44].

The Chapter is organized as follows. After discussing the role of the reactive navigation and related work, Section 2.3 explains the *situated-activity* design methodology. Section 2.4 introduces the reactive navigation method design. In Section 2.5 the *Nearness Diagram Navigation* method is implemented. Section 2.6 validates the research with experimentation on the real robot. Section 2.7 discusses the contributions of our approach regarding other methods, and in Section 2.8 the conclusions are drawn.

2.2 Preliminaries in Reactive Navigation

This Section discusses the application context where the *reactive navigation methods* have better performance than techniques such as *motion planning* (Subsection 2.2.1). Related work is analyzed in order to place our approach, and to present the motivation for this work (Subsection 2.2.2).

2.2.1 The Reactive Navigation Problem

There are many ways to generate collision-free motion. Roughly, they can be divided into those that are *global*, and based on *a priori* information (motion planning). And into those that are *local*, and based on *sensory information* (reactive navigation methods). Both techniques have some differences that justify their usage depending on the application constraints. The context of this work is to move a robot in unknown, unstructured, and dynamic scenarios.

Motion planning techniques calculate a collision-free path between the robot and goal configurations (see [35] for a review). From the path, motion commands are computed, and executed in real-time by the robot. The advantage of these algorithms is that they provide a global solution for reaching the goal. However, in unknown, unstructured, and dynamic scenarios the behavior of the motion planning algorithms is diminished. Dealing with sudden environmental changes, new sensory perceptions need to be integrated into a model, and continuous re-planning are required. Both tasks are time consuming, not complying with the real-time requirement. Besides, motion planning do not solve some situations (e.g. a dynamic obstacle temporally locates itself on top of the goal location). In these situations, the motion control loop cannot be closed, see Chapter 6.

The **reactive navigation methods** compute at each sample period one collision-free motion command, to converge the current robot location towards the goal location. The result of applying these algorithms is a sequence of motions that drive the robot free of collisions towards the goal location. One of the advantages is that explicit models of the environment are not required. Another advantage is to not impose an expensive computational load. The reactive methods are thus well-suited to deal with unknown, dynamic, and non-predictable scenarios. The drawback is that these methods use a local fraction of the information available (sensory information). Then, it is difficult to obtain optimal solutions, and to avoid the trap situations.

The attention focuses on reactive navigation methods as they fitting in the context of our application. The next Subsection presents related work to introduce the motivation for this work.

2.2.2 Related Work

The related work is introduced grouped according to the perception-action process carried out.

- There are methods that use a physical analogy to compute the motion commands. Some mathematical equations are applied to the sensory information. The solutions are transformed into motion commands (e.g. the potential field methods [30], [34], [58], [29], [9], [50], the perfume analogy [6], the fluid analogy [43], among others).

- There are methods that first compute some sets of motion commands. Next, a navigation strategy selects one motion command of these sets. Some methods calculate sets of steering angles [22], [11], [59], [28]. Others compute sets of velocity commands [56], [25], [13], [4].

- There are methods that calculate some form of high-level information description from the sensory information. Then, a motion command is computed, as opposed to being selected from a pre-calculated set. [54], and [29] formulate the concept of bubbles to describe parts of the free space. A collision-free path, included within the bubbles, is used to compute the motion commands. The *Nearness Diagram Navigation* method belongs to this group of approaches.

The **motivation for this work** is to develop a reactive navigation method to achieve safe navigation in *very dense, cluttered, and complex scenarios*: a task that presents a high degree of difficulty for most of the methods mentioned above.

2.3 The Situated-Activity Paradigm of Design

One of the goals of this Chapter is the design of a reactive navigation method using a classic design paradigm. This Section introduces the design paradigm selected, pointing out the advantages, difficulties, and requirements for its application.

The *situated-activity* is a paradigm for designing behaviors in Robotics [3]. The paradigm is based on defining a set of situations that describe the relative state of the problem entities. Subsequently, an action is designed for each situation. During the execution phase, perception is used to identify the current situation, and the associated action is carried out.

This paradigm has some **advantages** for designing modules that execute action-tasks with sensory information:

- The paradigm itself is a perception-action process.
- The paradigm utilizes a "divide and conquer" strategy to reduce the task difficulty.
- The paradigm does not suffer the real-time *action coordination problem*. This problem arises when the main task is divided into sub-tasks to reduce the task difficulty [1].

The **restriction** for the application of the paradigm is to find a set of situations that effectively describes the task-problem to solve (as commonly pointed out [26]).

The usage of this paradigm has to comply with some design **requirements**:

- The situations have to be *identifiable* from sensory perception, *exclusive*, and *complete* to represent the relative state of the problem entities. Moreover, the *explosion in the number of situations* needed has to be avoided.
- Each action design has to solve *individually* the task problem, in the context of each situation.

2.4 The Reactive Navigation Method Design

The goal of this Section is to understand the design of a reactive navigation method using the *situated-activity* paradigm. The Section presents the design, and analyzes the completion with the paradigm requirements.

Fig. 2.2 depicts the reactive navigation method design that works as follows: at each sample period, the sensory information is used to identify the current situation among the predefined set (introduced in Subsection 2.4.1). Subsequently, the associated action is executed computing the motion. Subsection 2.4.2 addresses the action design.

[1] In short, when a real-time task is decomposed in some sub-tasks, a decision algorithm might arbiter them - to decide which sub-task is active in real-time. This is the action coordination problem.

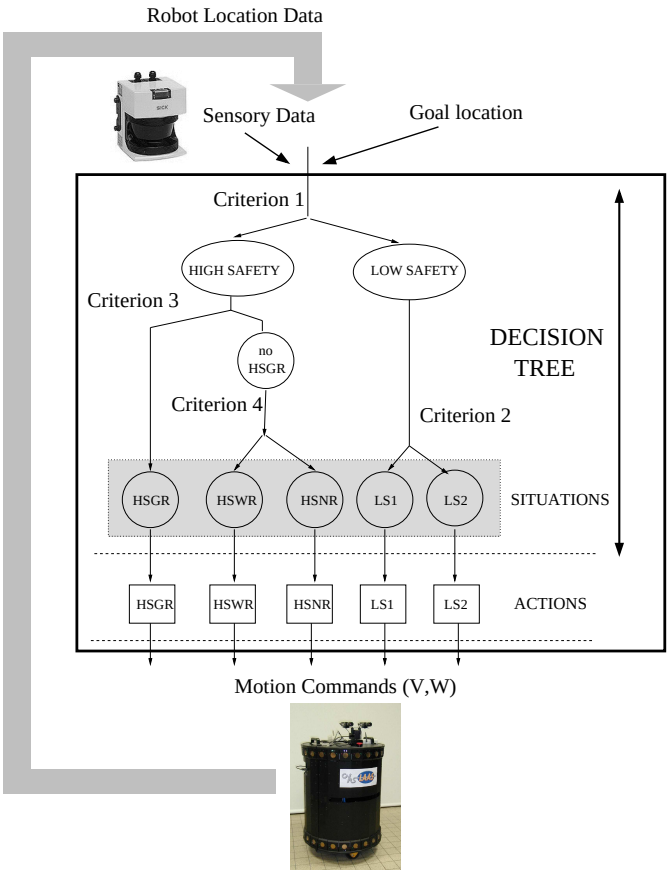


Figure 2.2: Reactive navigation method design.

2.4.1 General Situations Definition

The general situations describe the relative state of the reactive navigation entities: the robot, the obstacle distribution, and the goal location. The relations among them are studied below.

The Robot-Goal-Obstacle Relations

The robot and obstacle distribution relation
The relation between the robot and the obstacle distribution is analyzed with a *security distance*, that creates a *security zone* around the robot bounds, see Fig. 2.3a. The robot and obstacles relation is analysed with a safety evaluation.

The robot and goal relation

The relation between the robot and goal location is obtained with an intermediate device, the *free walking area*. To compute the *free walking area*, *gaps* in the obstacle distribution are searched for. Two contiguous *gaps* form a *region*. Finally, the angular closest *region* to the goal location is selected. It must be checked whether this region is "navigable" by the robot. This *region* is called *free walking area*, see Fig. 2.3a. The robot and goal relation is obtained from the obstacle distribution structure with the *free walking area*.

Situations Definition and Representation

A *decision tree* is used to represent the situations, see Fig. 2.2. The tree inputs are the robot and goal locations, and the obstacle distribution (sensory information). The tree output is the current situation. The tree is traversed using binary decision rules, that are based on criteria defined over the entities. The four criteria are listed below.

Criterion 1: Safety criterion. There are two safety situations: Low Safety (LS) and High Safety (HS), see Fig. 2.2. The robot is in Low Safety if there are obstacles within the *security zone*. The obstacles present a risk for the robot, Figs. 2.3a,b. If not, the situation is High Safety, Figs. 2.3c,d,e.

There are two Low Safety situations, see Fig. 2.2. The first two general situations are obtained by applying the following criterion:

Criterion 2: Dangerous obstacle distribution criterion.

1. **Low Safety 1** (LS1): The robot is in LS1 when there are obstacles within the *security zone*, but only on one side of the *free walking area gap*, which is the closest to the goal, see Fig. 2.3a.

2. **Low Safety 2** (LS2): The robot is in LS2 when there are obstacles within the *security zone*, on both sides of the *free walking area gap*, which is the closest to the goal see Fig. 2.3b.

There are three High Safety situations, see Fig. 2.2. The third general situation is obtained by applying the following criterion.

Criterion 3: Goal and *free walking area* criterion.

3. **High Safety Goal in Region** (HSGR): The robot is in HSGR if the goal location is within the *free walking area*, see Fig. 2.3c.

If not, the fourth and fifth general situations are obtained by applying the last criterion.

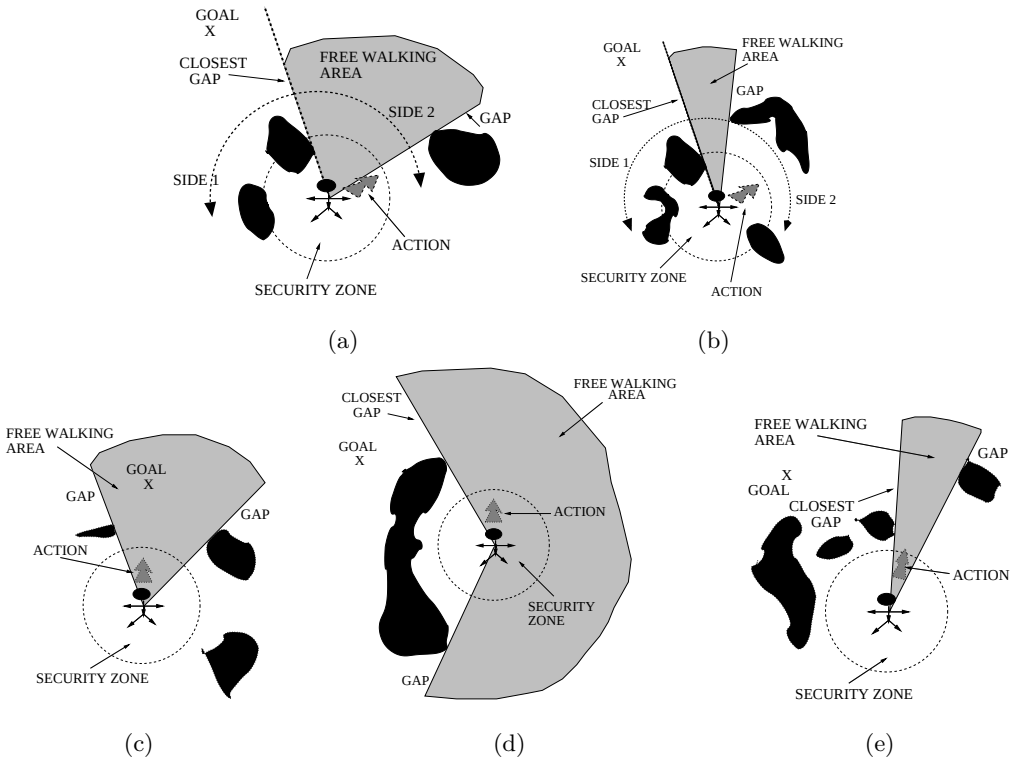


Figure 2.3: a) LS1 situation/action example. b) LS2 situation/action example. c) HSGR situation/action example. d) HSWR situation/action example. e)HSNR situation/action example.

Criterion 4: *Free walking area* width criterion. A *free walking area* is *wide* if its angular width is larger than a given angle. If not, the *free walking area* is *narrow*.

4. **High Safety Wide Region** (HSWR): The robot is in HSWR if the goal is not within *the free walking area*, but the *free walking area* is *wide*, see Fig. 2.3d.

5. **High Safety Narrow Valley** (HSNR): The robot is in HSNR if the goal is not within the *free walking area*, but the *free walking area* is *narrow*, see Fig. 2.3e.

Some conclusions regarding the definition and representation of these situations are summarized next:

- The situations are *identifiable* from sensory perception. (When the sensory information is available as depth maps.)
- The situations are *exclusive* and *complete*. They are obtained from a binary *decision tree*.
- The situation definition avoids the *explosion in the number of situations*, because there are five. This comes from the fact that, the situation definition does not depend on the resolution or size of the space considered.

Then, the definition and representation of the situations comply with the requirements imposed by the *situated-activity* paradigm, mentioned in Section 2.3.

2.4.2 Action Design

This Subsection presents the action design guidelines:

1. **Low Safety 1** (LS1): This action moves the robot away from the closest obstacle, while directing the robot towards the *free walking area gap* closest to the goal, see Fig. 2.3a.
2. **Low Safety 2** (LS2): This action keeps the robot at the same distance from the two closest obstacles, while moving the robot towards the *free walking area gap* closest to the goal, see Fig. 2.3b.
3. **High Safety Goal in Region** (HSGR): This action drives the robot directly to the goal, see Fig. 2.3c.
4. **High Safety Wide Region** (HSWR): This action moves the robot alongside the obstacle, see Fig. 2.3d.
5. **High Safety Narrow Region** (HSNR): This action directs the robot through the central zone of the *free walking area*, see Fig. 2.3e.

Conclusions regarding the action design are next summarized. Each action solves *individually* the task problem, in the context of each situation: Avoiding obstacles while moving the robot towards the goal location. This is achieved in Low Safety and High Safety actions as follows:

- In Low Safety, both actions avoid the obstacles, while moving the robot towards the *free walking area gap* closest to the goal. This *gap* implicitly has information about the goal location.

- In High Safety, the actions drive the robot towards the goal, towards the *free walking area gap* closest to the goal, or towards the central zone of the *free walking area*. Then, these actions explicitly or implicitly drive the robot towards the goal location. The robot is not in danger of collision, so there is no need to avoid obstacles.

Then, the actions design comply with the requirements imposed by the *situated-activity* paradigm, mentioned in Section 2.3.

There are some points worth mentioning here:

1. The reactive navigation method design is described in symbolic level. Different implementations of this design lead to new reactive navigation methods. Learning techniques, Fuzzy sets, Potential Field implementations, optimization techniques and other tools may be used to implement the design. The next Section presents a geometry-based implementation.

2. Any reactive navigation method implemented following the proposed design simplifies the reactive navigation problem (by a "divide and conquer" strategy based on situations). So, a good implementation might solve more difficult navigation problems than other existing methods, that use a unique navigation heuristic, do. Besides, the design is flexible. New situations might be defined to reduce the difficulty even more.

3. The design does not suffer from the "action coordination problem". The sub-tasks are self-coordinated, because the general situations are *complete* and *exclusive*. Only one situation can be selected at each time, and thus, only one action is executed.

Summarizing, this Section has presented the design of a reactive navigation method using the *situated-activity* paradigm. The Section also demonstrated that the situations, and the associated actions, comply with the requirements imposed by the paradigm.

2.5 The Nearness Diagram Navigation (ND)

This Section presents a geometry-based implementation of the design called the *Nearness Diagram Navigation*.

The attention is focused on a circular (with radius R) holonomic robot moving over a flat surface. The Workspace $\mathcal{W}$ is $\mathbb{R}^2$. A point $\mathbf{x} = (x, y) \in \mathbb{R}^2$ is a location of the robot. The space of motion commands is three-dimensional.

Let $(\mathbf{v}, \mathbf{w})$ be a motion command. Let $\mathbf{v} = (v_m, \theta)$ be the translational velocity, and let $\mathbf{w}$ be the rotational velocity.

The sensory information is supposed to be depth point maps for two reasons: (1) maintaining the sensor as general as possible - the great majority of sensory information can be processed and then reduced to points, and (2) avoiding the use of structured information (as lines, polygons, etc), that otherwise can be used if it is available. Let L be a list of the N obstacle points perceived at each time.

Fig. 2.2 depicts the ND method design that works as follows: at each sample period T the sensory information is used to identify the current situation among the predefined set (introduced in Subsection 2.5.1). Subsequently, the associated action is executed computing the motion commands, $(\mathbf{v}, \mathbf{w})$. The actions are control laws presented in Subsection 2.5.2.

2.5.1 Information Representation and General Situations

This Subsection introduces the tools used to analyze the sensory information: the Nearness Diagrams. Subsequently, the robot location, obstacle distribution, and goal location relations are used to define the general situations.

Nearness Diagram Definition

The ND divides the space in sectors whose centre is over the robot centre. Let n be the number of sectors ($n = 144$ in our implementation, so $2.5°$ is the angle of each sector). Notice that in the robot reference, the bisector of the $\frac{n}{2}$ sector is $0°$, of the $\frac{n}{4}$ is $\frac{\pi}{2}$, and of the $\frac{3n}{4}$ is $\frac{-\pi}{2}$.

Let $\delta_i(\mathbf{x}, L)$ be the function that computes the distance from the robot centre to the closest obstacle point in sector i. $\delta_i(\mathbf{x}, L) = 0$ when there are no obstacles in sector i. $max(\delta_i(\mathbf{x}, L)) = d_{max}$, where d_{max} is the maximum range of the sensor. Then, let $D = \{d_i = \delta_i(\mathbf{x}, L), i = 1 \dots n\}$ be the list of the minimum obstacle distances to the robot centre in each sector.

Definition 1 *Nearness Diagram from the central Point (PND)*

$$\begin{array}{rrcl} PND: & \mathbb{R}^2 \times D & \rightarrow & \{\mathbb{R}^+ \cup \{0\}\}^n \\ & (\mathbf{x}, d_i) & \rightarrow & \{PND_i(\mathbf{x}, d_i)\}^n \end{array}$$

$$\begin{array}{ll} \textit{if } d_i > 0, & PND_i(\mathbf{x}, d_i) = d_{max} + 2R - d_i \\ \textit{else} & PND_i(\mathbf{x}, d_i) = 0 \end{array}$$

Definition 2 *Nearness Diagram from the Robot bounds (RND)*

$$\begin{aligned} RND: \quad \mathbb{R}^2 \times D &\rightarrow \{\mathbb{R}^+ \cup \{0\}\}^n \\ (\mathbf{x}, d_i) &\rightarrow \{RND_i(\mathbf{x}, d_i)\}^n \end{aligned}$$

$$\begin{aligned} &if\, d_i > 0, \quad RND_i(\mathbf{x}, d_i) = d_{max} + E_i - d_i \\ &else \qquad\quad RND_i(\mathbf{x}, d_i) = 0 \end{aligned}$$

where

- E*: function that depends on the robot geometry. The function E_i is the robot radius, R, for a circular robot* [2].

The PND represents the nearness of the obstacles from the robot centre. The RND represents the nearness of the obstacles from the robot boundary. Some PND and RND diagrams are illustrated in Fig. 2.4. From now on, $PND_i \equiv PND_i(\mathbf{x}, d_i)$ and $RND_i \equiv RND_i(\mathbf{x}, d_i)$.

The Robot-Goal-Obstacle Relations

The relations among the robot, the obstacle distribution, and the goal are analyzed by using these diagrams.

The robot and obstacle distribution relation

The relation between the robot and the obstacle distribution is obtained with a safety evaluation. This evaluation is made with the RND, because it represents the nearness of the obstacles from the robot bounds. Let the *security distance* be the minimum obstacle admissible distance, d_s, to the robot bounds. A *security nearness* is computed by $n_s = d_{max} - d_s$, and used in the RND to evaluate the robot safety, see Fig. 2.4c.

The robot and goal relation

The relation between the robot and the goal location is obtained from a high level device called *free walking area*. It is computed from the PND, since it represents the nearness from the obstacles to the robot centre. The PND analysis is carried out in three stages. First, *gaps* are identified. From these *gaps*, *regions* are obtained, and finally one *region* is selected: the *free walking area*, see Fig. 2.4a.

1. *Gaps*: First, *gaps* in the obstacle distribution are identified searching for *discontinuities* in the PND.

[2] If the robot is not circular, E_i is the distance from the robot centre to the robot bounds in sector i.

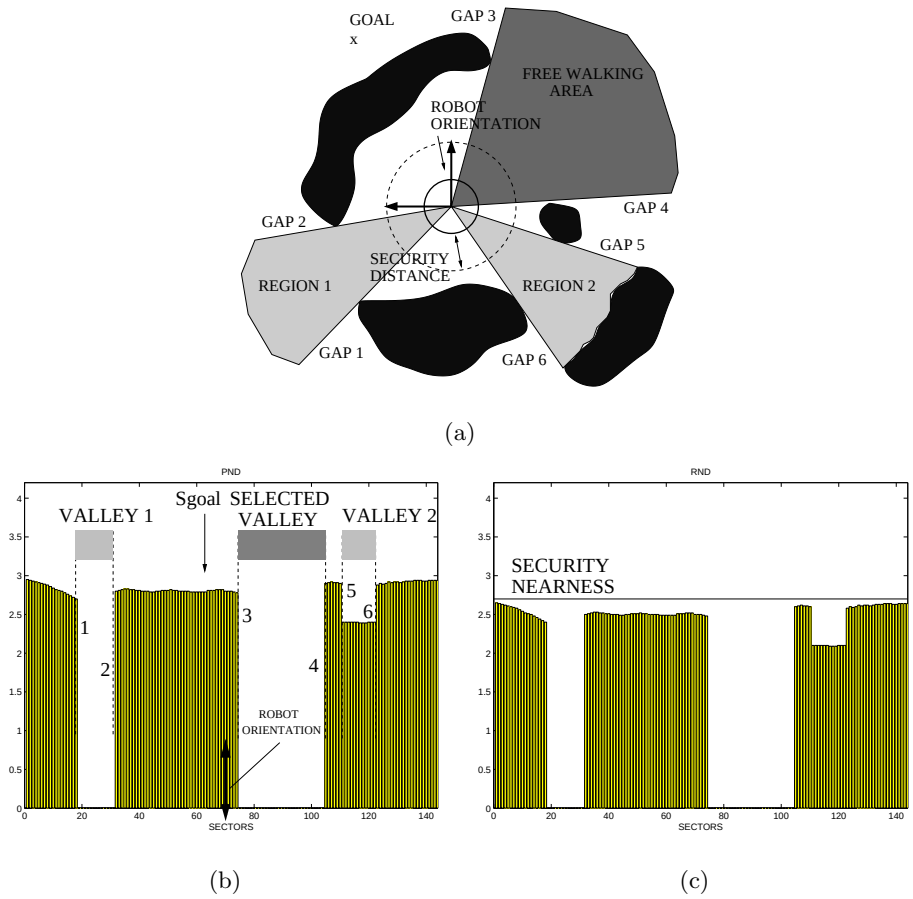


Figure 2.4: a) *Gaps*, *regions*, and *free walking area*. b) PND. c) RND. The following values were set: $R = 0.3m$, $d_{max} = 3m$, $d_s = 0.3m$.

A *discontinuity* exists between two sectors (i, j) if they are adjacent[3], and their PND value differs by more than $2R$ (robot diameter), that is $|PND_i - PND_j| > 2R$. Fig. 2.4a depicts the *gaps* searched for. They are identified by *discontinuities* in the PND, see Fig. 2.4b. Notice that the robot diameter is used because only gaps where the robot fits are searched for.

Two types of *discontinuities* are distinguished. When there is a *discon-*

[3]The adjacent sectors to i are $i - 1$ and $i + 1$. It is assumed that in all the operations among sectors the $mod(*, n)$ function is used. Thus, if $i = n$ then $i = 0$. This gives continuity to the diagrams.

tinuity between two sectors (i, j), then: if $PND_i > PND_j$ the *discontinuity* is a *rising discontinuity* from j to i, and a *descending discontinuity* from i to j.

Regions: Two contiguous *gaps* form a *region*. *Regions* are identified searching for *valleys* in the PND.

Let be a *valley* a non empty set of sectors, $S = \{i + p\}_{p=0,\cdots,r}$ with $n > r \geq 0$, that satisfies the following conditions:

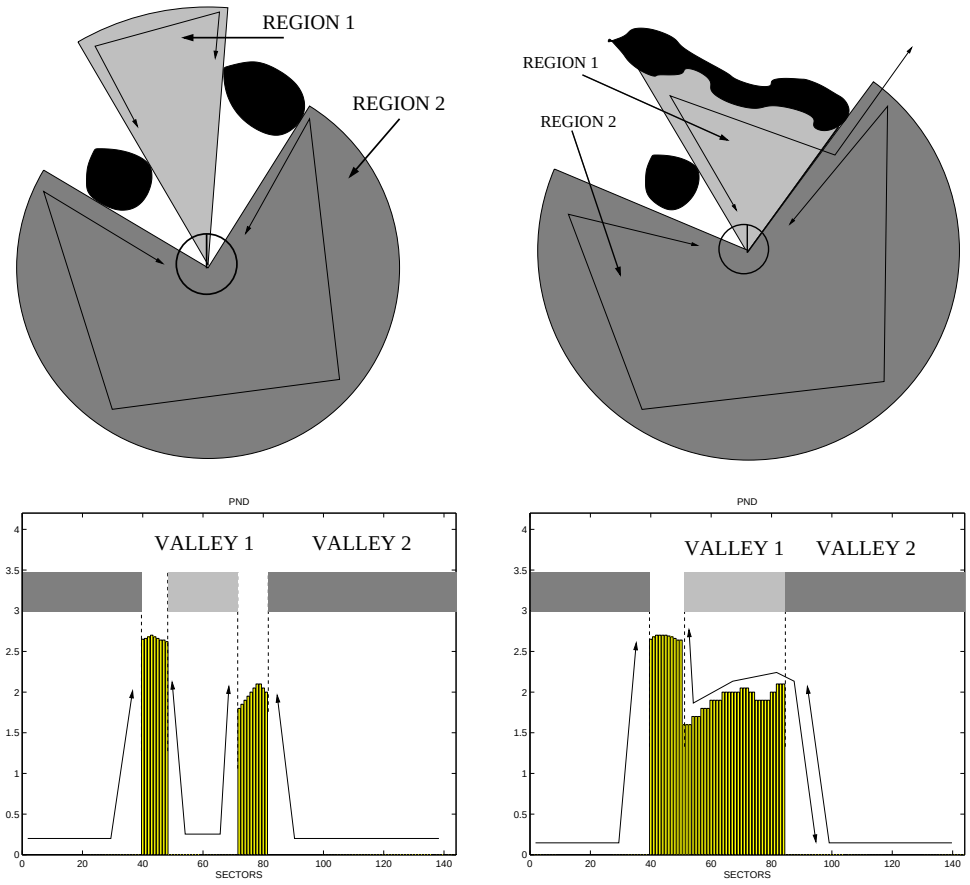
(a) There are no *discontinuities* between adjacent sectors of S.

(b) There exist two *discontinuities*:
$|PND_{i-1} - PND_i| > 2R \quad AND \quad |PND_{i+p} - PND_{i+p+1}| > 2R$

(c) One of these *discontinuities* is a *rising discontinuity* from i to $i-1$ or from $i+p$ to $i+p+1$:
$PND_{i-1} > PND_i \quad OR \quad PND_{i+p+1} > PND_{i+p}$

Fig. 2.5 depicts some *regions* and the *valleys* identified in the PND. There are not *gaps* within a *region*. Then, there cannot be *discontinuities* within a *valley* (condition (a)). A *region* has one *gap* at each side. Then, a *valley* has two *discontinuities* in the extreme sectors i and $i+p$, $(i-1, i)$ and $(i+p, i+p+1)$. The sectors $i-1$ and $i+p+1$ do not belong to the *valley*, but are adjacent to i and $i+p$ respectively (condition (b)). Finally, at least one of the *discontinuities* of the *valley* has to be *rising* (condition (c)). Fig. 2.5a depicts two *regions* that are identified by *valleys* created by two *rising discontinuities*. Valley1 in Fig. 2.5b is created by one *descending* and one *rising discontinuity*. Thus, it identifies the region1. The number of *rising discontinuities* structurally differs the two types of *regions* considered (e.g. in Fig. 2.5b, region2 is identified by valley2 that has two *rising discontinuities*, however region1 is identified by valley1 that has only one *rising discontinuity*).

A special case occurs when the goal is between an obstacle and the robot. The sector that contains the goal location, s_{goal}, could not belong to a *valley*. However, it is desirable that when this condition arises, the goal was within a *region*. Thus, when this situation is detected, the $PND_{s_{goal}}$ value is set equal to zero. This creates an artificial *valley* in the goal sector, that is, the goal is within a *region*. This case is illustrated in Fig. 2.6. Another special case is when there are not obstacles. Then all the sectors form the *valley*.



(a) (b)

Figure 2.5: Types of regions

2. *Free walking area*: The *free walking area* is the "navigable" *region* closest to the goal location. To identify it, the *valley* with the *rising discontinuity* closest[4] to s_{goal} is selected (see Fig 2.4, the *valley* created by *discontinuities* 3 and 4 is selected, because *discontinuity* 3 is the *rising discontinuity* closer to s_{goal}). Then, it is checked whether the candidate *region* is "navigable" (see Appendix A for a description of the algorithm). If it is not "navigable", another *valley* is selected and the process is repeated until a "navigable" *region* is found or no *region* exists. The selected

[4]The term closest is in number of sectors.

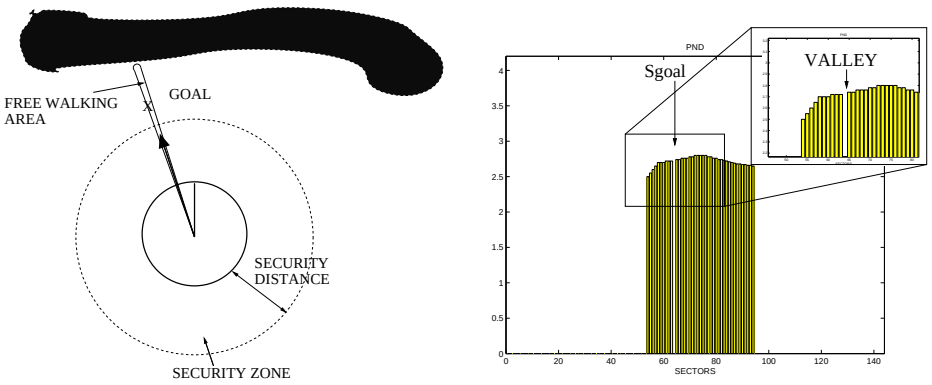


Figure 2.6: HSGR situation/action example. The artificial *valley* case.

valley identifies the *free walking area*, see Fig. 2.4.

A *free walking area* is *wide* if its angular width is greater than a given quantity (in our current implementation it is considered 90°). If not, the *free walking area* is *narrow*. Then, a *valley* is *wide* if the number of sectors is greater than $s_{max} = \frac{n}{4}$. If not, the *valley* is *narrow*. Notice that the number of sectors of a *valley* is the angular width of the *region*.

General Situations

The general situations definition correspond to the reactive method situations mentioned in Section 2.4.1. The situations are represented in the same *decision tree* presented in Fig.2.2. The criteria to traverse the tree branches in order to identify the situations are listed below.

Criterion 1: Safety criterion. To evaluate the robot safety the *security nearness* is used. The safety situation is Low Safety (LS) if there are obstacles that exceed the *security nearness* in the RND. If not, the situation is High Safety (HS), see Fig. 2.8.

There are two Low Safety situations, see Fig. 2.2. They are obtained by applying the following criterion:

Criterion 2: Dangerous obstacle distribution criterion.

1. **Low Safety 1** (LS1): The robot is in LS1 if there is, at least, one sector that exceeds the *security nearness* in the RND, only on one side of the selected *valley rising discontinuity* which is closest to the goal sector.

   See Fig. 2.8. The *rising discontinuity* of the selected *valley* closest to the

goal sector (s_{goal}), is s_i (see the PND). Then, there are sectors exceeding the *security nearness* only on one side of s_i in the RND .

2. **Low Safety 2** (LS2): The robot is in LS2 if there is, at least, one sector that exceeds the *security nearness* in the RND, on both sides of the selected *valley rising discontinuity* which is closest to the goal sector.

   See Fig. 2.8. The *rising discontinuity* of the selected *valley* closest to the goal sector (s_{goal}), is s_i (see the PND). Then, there are sectors exceeding the *security nearness* on both sides of s_i in the RND.

There are three High Safety situations, see Fig. 2.2. The third general situation is obtained by applying the following criterion:

Criterion 3: Goal and *free walking area* criterion.

3. **High Safety Goal in Region (HSGR)**: The robot is in HSGR if the goal sector belongs to the selected *valley*.

   See Fig. 2.8. The goal sector (s_{goal}) belongs to the selected *valley*, see the PND. Notice that no sector exceeds the *security nearness* in the RND.

If not, the fourth and fifth general situations are obtained by applying the last criterion.

Criterion 4: *Free walking area* width criterion.

4. **High Safety Wide Region (HSWR)**: The robot is in HSWR if the goal sector does not belong to the selected *valley*, but the *valley* is *wide*.

   See Fig. 2.8. The goal sector (s_{goal}) do not belong to the selected *valley* that is *wide* (the number of sectors of the *valley* is $102 > \frac{n}{4} = 36$), see the PND. Notice that no sector exceeds the *security nearness* in the RND.

5. **High Safety Narrow Region (HSNR)**: The robot is in HSNR if the goal sector does not belong to the selected *valley*, but the *valley* is *narrow*.

   See Fig. 2.8. The goal sector (s_{goal}) do not belong to the selected *valley* that is *narrow* (the number of sectors of the *valley* is $13 < \frac{n}{4} = 36$), see the PND. Notice that no sector exceeds the *security nearness* in the RND.

This Section has described the tools used to represent the sensory information, and the definition and identification of the general situations.

2.5.2 Associated Actions

This Subsection introduces the implementation of the actions associated with the general situations. The objective is to find simple control laws that produce the desired behavior associated with the situation identified (following the reactive method design, see Section 2.4.2). The actions compute the motion commands $(v_m, \theta, \mathbf{w})$. The action implementation is summarized in Table *I*.

Translational Velocity Direction (θ)

A solution sector, $s_\theta \in \mathbb{R}$, is calculated for each situation. The direction of motion θ (translational velocity direction) is the bisector of s_θ. As $s_\theta \in \mathbb{R}$, any direction of motion can be assigned $\theta \in [-\pi, \pi]$. For a realistic implementation of the method, instantaneous backwards motion is prohibited ($\theta \in [-\pi/2, \pi/2]$, $s_\theta \in [\frac{n}{4}, \frac{3*n}{4}]$).

Translational velocity direction in Low Safety

In Low Safety the robot is in danger of colliding because there are obstacles within the *security zone*. The robot must be brought to a secure situation.

1. **Low Safety 1**: *In LS1, the objective is to move the robot away from the closest obstacle, while directing the robot towards the free walking area gap closest to the goal*, see Fig. 2.8. The solution sector, s_θ, is calculated by:

$$\begin{aligned} s_p &= |s_i - s_j| * p + \tfrac{s_{max}}{2} \\ s_\theta &= s_i + sign(s_i - s_j) * s_p \end{aligned} \tag{2.1}$$

   where:

   - s_i: sector corresponding to the selected *valley rising discontinuity* which is closest to the goal sector (s_{goal}). This sector contains the *free walking area gap* closest to the goal.
   - s_j: sector with the highest value in the RND, that exceeds the security nearness, on one side of s_i. This sector contains the closest obstacle point.
   - p: experimentally tuned parameter. Its value depends on the transitions among the general situations, to ensure a smooth behavior among them. The parameter acts as an adaptable proportional controller. In the current implementation $p \in [1.5, 2.5]$.

TABLE:	*ACTIONS*		
SITUATIONS	θ Trans. vel. angle	v_m Trans. vel. module	**w** Rot. vel.
LS1	$s_p = \lvert s_i - s_j \rvert * p + \frac{s_{max}}{2}$ $\theta = bisec(s_\theta = s_i + sign(s_i - s_j) * s_p)$	$v_m = v_{max} * \frac{d_{obs}}{d_s} * (\frac{\frac{\pi}{2} - \lvert\theta\rvert}{\frac{\pi}{2}})$	$\mathbf{w} = w_{max} * \frac{\theta}{\frac{\pi}{2}}$
LS2	$s_{med_1} = \frac{s_m + s_j}{2}$ $s_{med_2} = \frac{s_m + s_j + n}{2}$ If $\lvert s_i - s_{med_1} \rvert < \lvert s_i - s_{med_2} \rvert$ Then $\theta = bisec(s_\theta = s_{med_1} \pm c)$ Else $\theta = bisec(s_\theta = s_{med_2} \pm c)$		
HSGR	$\theta = bisec(s_\theta = s_{goal})$	$v_m = v_{max} * (\frac{\frac{\pi}{2} - \lvert\theta\rvert}{\frac{\pi}{2}})$	
HSWR	$\theta = bisec(s_\theta = s_i \pm \frac{s_{max}}{2})$		
HSNR	$\theta = bisec(s_\theta = \frac{s_i + s_j}{2})$		

Table 2.1: Situation/Action table.

2. **Low Safety 2**:*In LS2, the objective is to keep the robot at the same distance from the two closest obstacles, while moving the robot towards the free walking area gap closest to the goal*, see Fig. 2.8. The solution sector is computed by:

$$s_{med_1} = \frac{s_m + s_j}{2} \quad s_{med_2} = \frac{s_m + s_j + n}{2}$$

$$If \;\; |s_i - s_{med_1}| < |s_i - s_{med_2}| \;\; then \;\; s_\theta = s_{med_1} \pm c$$

$$Else \;\; s_\theta = s_{med_2} \pm c \tag{2.2}$$

where:

- s_i: sector corresponding to the selected *valley rising discontinuity* which is closest to the goal sector (s_{goal}). This sector contains the *free walking area gap* closest to the goal.
- s_m, s_j: the sectors with the highest values in the RND, that exceed the *security nearness*, on both sides of s_i. These sectors contain the two closest obstacles on both sides of the *free walking area*.
- c: is a correction function used to keep the robot centered between the two closest obstacles. The function depends on the closest obstacle distance, and on the difference between the distances of the

two closest obstacles. This quantity, c, is added or subtracted due to the sector that contains closest obstacle.

Translational velocity direction in High Safety

In High Safety the robot is not in danger of colliding. Thus, the robot moves within the free walking area.

3. **High Safety Goal in Region**: *In HSGR the objective is to drive the robot directly to the goal*, see Fig. 2.8. The solution sector is calculated by $s_\theta = s_{goal}$.

   The goal location is explicitly used to compute the motion commands only in this situation. Notice that the robot is not in danger of colliding, and the goal is within the *free walking area*. This situation is not dangerous for the robot, and does not appear to exhibit complexity. Fig. 2.6 depicts the special case where an artificial *valley* is created preserving the HSGR situation.

4. **High Safety Wide Region**: *In HSWR the objective is to cause a motion alongside the obstacle*, see Fig. 2.8. The solution sector is calculated by:

$$s_\theta = s_i \pm \frac{s_{max}}{2} \tag{2.3}$$

   where:

   - s_i: sector corresponding to the selected *valley rising discontinuity* which is closest to the goal sector (s_{goal}). This sector contains the *free walking area gap* closest to the goal.

5. **High Safety Narrow Region**: *In HSNR the objective is to direct the robot through the central zone of the free walking area*, see Fig. 2.8.

$$s_\theta = \frac{s_i + s_j}{2} \tag{2.4}$$

   where:

   - s_i, s_j: sectors of the two *discontinuities* of the selected *valley*. These sectors contain the two *gaps* of the *free walking area*.

Translational Velocity Absolute Value (v_m)

The translational velocity value, v_m, is computed depending on whether the robot is in High Safety or Low Safety. Let v_{max} be the maximum translational velocity. Let d_{obs} be the distance from the closest obstacle to the robot bounds, and let d_s be the *security distance*. Let $\theta \in [-\pi/2, \pi/2]$ be the translational velocity direction calculated. Then:

1. High Safety:

$$v = v_{max} * (\frac{\frac{\pi}{2} - |\theta|}{\frac{\pi}{2}}) \tag{2.5}$$

2. Low Safety:

$$v = v_{max} * \frac{d_{obs}}{d_s} * (\frac{\frac{\pi}{2} - |\theta|}{\frac{\pi}{2}}) \tag{2.6}$$

With this velocity control, the robot moves at maximum speed until one obstacle shows up in the *security zone*. Then, the robot reduces the speed in proportion to the distance to the closest obstacle, until the *security zone* is clear. Moreover, sudden changes in the direction of motion reduce the translational velocity module.

Rotational Velocity (w)

This velocity term is required when the sensor has visibility constraints. It aligns the main sensor direction with the robot instantaneous direction of motion (it is considered for the following analysis that the main sensor direction and the robot orientation match).

The rotational velocity, $\mathbf{w}$, is computed from the translational velocity direction, θ. The robot must be aligned with the instantaneous direction of motion. Let w_{max} be the maximum rotational velocity. Then:

$$\mathbf{w} \quad = w_{max} * \frac{\theta}{\frac{\pi}{2}} \tag{2.7}$$

This produces sudden turns of the robot when there are great changes in θ (the robot rotates facing the direction of motion as soon as possible), and smooth turns when the changes are small.

Summarizing, in this Section the ND method has been presented. The ND method is a geometry-based implementation of the reactive method design. The reactive navigation method computes from the sensory information, the motion commands $(v_m, \theta, \mathbf{w})$, to safely drive a vehicle among locations.

2.6 Implementation and Experimental Results

The goal of this Section is to experimentally validate the ND method.

2.6.1 The Mobile Platform

The ND method has been tested on a *Nomadic XR4000* at *LAAS-CNRS*, France. The robot is equipped with a *SICK* 2-D laser rangefinder. For further details about the platform and sensor see Appendix E. The sample period of the ND method is around $125msec$ on the on-board *Pentium II*. In the current implementation, a short-time memory built with the last 20 laser measurements (361×20 points) is used. The maximum translational velocity is set to $v_{max} = 0.3\frac{m}{sec}$, and the maximum rotational one is set to $w_{max} = 1.57\frac{rad}{sec}$. These velocity limits were selected due to the potential applications of the method: safe motion in indoor human environments. These scenarios exhibit a high density of obstacles and the robot must work with humans around preserving their safety, see Fig. 2.1.

2.6.2 Experimental Results

Three experiments using the real platform are discussed next. In all the experiments the scenario was unknown. The environment might be unstructured, dynamic, and non predictable. Only the goal location was available in advance. These circumstances justify the usage of a reactive navigation method to move the robot, see Subsection 2.2.1.

The experiments were designed to verify that the ND method complies with the motivation of this work: *to safely drive a robot in very dense, cluttered, and complex scenarios.* The experiments will also allow for discussion of the method technical contributions in the next Section, which are summarized as follows: (1) avoiding trap situations due to the perceived environment structure (e.g. U-shape obstacles, two very close obstacles); (2) generating stable and oscillation-free motion; (3) selecting motion directions towards obstacles; (4) exhibiting a high goal insensitivity - i.e. to be able to choose motion directions far away from the goal direction; (5) selecting regions of motion using a robust "navigability" criterion.

For each experiment an upper view is shown: the robot trajectory and all the laser points perceived, see Fig. 2.9a. The robot velocity profiles are also illustrated, see Fig. 2.9b. Some snapshots have been selected for better understanding of the robot motion - the floor tiles are 10cm square and the robot

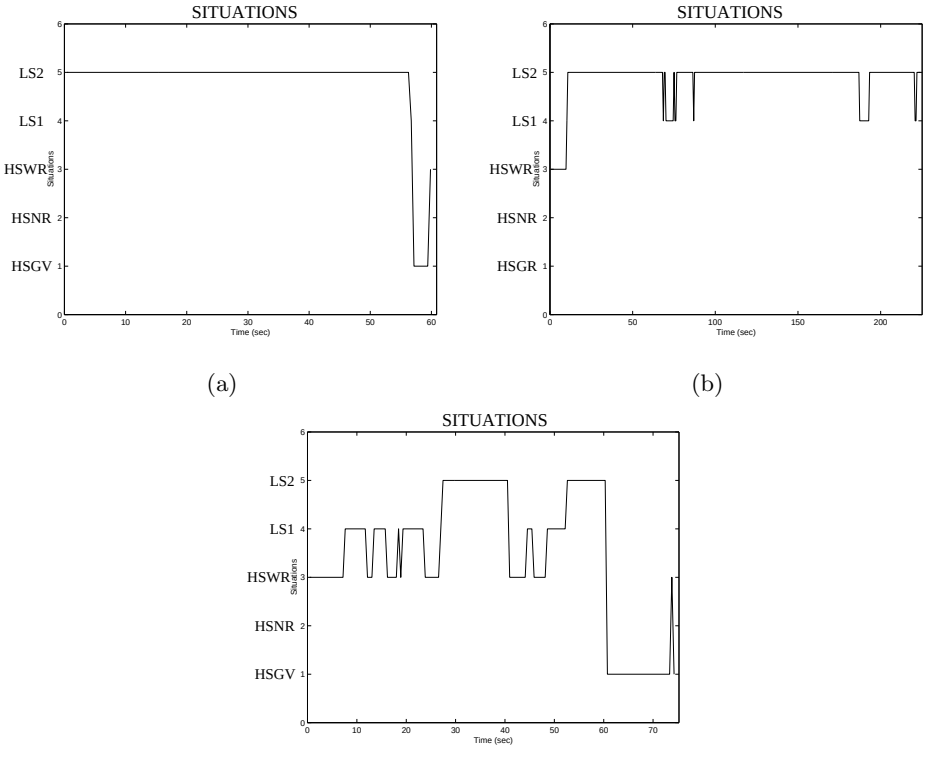


Figure 2.7: Situations during the experiments. 1: HSGV. 2: HSNV. 3: HSWV. 4: LS1. 5: LS2. a) Experiment 1. b) Experiment 2. c) Experiment 3.

diameter is 48cm, see Fig. 2.9d. Moreover, at some selected times (matching with some snapshots), the robot location, the perceived laser points, and the method direction solution are shown, see Fig. 2.9k. The robot situation selected at each time is illustrated in Fig. 2.7.

- **Experiment 1**: This experiment was designed to test the robot moving in narrow places with highly reduced room to maneuver. In order to reach the goal location, the robot traversed a very dense, complex, and cluttered passage, see Fig. 2.9a.

  In some parts of the experiment the robot was required to move among very close obstacles $< 10cm$ on both sides (notice that the tiles size is about $10cm$), see Figs. 2.9d,f,g. No trap situations appeared due to the motion in narrow places.

The robot entered and traveled along the passage, because the possibility of whether the passage was "navigable" was checked every time by the *free walking area*. The *free walking areas* selected at some instants are illustrated in Figs. 2.9e-l,f-m.

The selection of directions towards the obstacles was essential to successfully accomplish this experiment. Figs. 2.9e-l,f-m,g-n depict some instants when the computed direction solution pointed towards an obstacle.

The motion computed and executed was oscillation-free. This is seen in the robot path made and in the velocity profiles, see Figs. 2.9a,b. The experiment was carried out in $60sec$, and the average translational velocity was $0.114\frac{m}{sec}$.

The Fig. 2.7a shows the situation selected at each time. Mainly, the robot was in Low Safety 2, because there were obstacles within the *security zone* on both sides of the *free walking area* at every moment.

- **Experiment 2**: This experiment tested the robot navigating in a dynamic environment. In order to reach the goal location, the robot moved in a dense, complex, and cluttered scenario, see Fig. 2.10a, that was dynamically built by a human while the robot was moving (Figs. 2.10d,h,i,k show the dynamic nature of the environment).

  In the first part of the experiment, the human closed the passage when the robot was in the first corridor, see Figs. 2.10f-p. The robot detected that it was trapped and it stopped (in Fig. 2.10b the velocities from second 33 to 52 are zero). Notice that a flag could be launched to a higher-level module to plan a new subgoal (but these experiments only test the reactive method). Finally, the passage was opened and the robot resumed the motion, see Fig. 2.10g.

  In some parts of the experiment, the robot moved among very close obstacles, see Figs. 2.10e-o. No trap situations were detected due to the motion in narrow places.

  The robot used the *free walking area* to select areas of motion with a width checking, see Fig. 2.10e-o,i-r.

  The method selected motion directions towards the obstacles when required (in almost all the experiment). Some of them are illustrated in Figs. 2.10e-o,h-q,i-r.

  To successfully navigate in this environment, the robot was required to

select directions of motion far away from the goal direction (mentioned before as goal insensitivity). This property is illustrated in Figs. 2.10h-q,i-r. Motion direction solutions and goal directions differ in more than 90° (any difference could be obtained with the reactive method).

No oscillations appeared during the run. This is seen in the robot path made and in the velocity profile, see Figs. 2.10a,b. The complete time of the experiment was $220sec$, and the average translational velocity was $0.104\frac{m}{sec}$.

The situation selected at each time is illustrated in Fig. 2.7b. The robot mainly is in Low Safety 2, since there were obstacles within the *security zone* on both sides of the *free walking area* at every moment. Sometimes the robot is in Low Safety 1, because there were risky obstacles only on one side of the *free walking area*. The reasons are the sensor visibility constraints, and the limited short-time memory. Sometimes, the robot did not "see" any obstacle on one side. However, when the robot turned, it could "see" the obstacles and the situation turned to be Low Safety 2.

- **Experiment 3**: This experiment was designed to test the trap situation avoidance. In order to reach the goal location, the robot avoided three U-shape obstacles placed in the environment, see Fig. 2.11a and Figs. 2.11c,g.

  The robot avoided entering and getting trapped because the U-shape obstacles were completely visible by the robot. The method uses the *free walking area* device to avoid these structural trap situations. The Figs. 2.11c-k,e-l,f-m,g-n depicts some parts of the experiment and the *free walking areas* selected. The *free walking areas* of Figs. 2.11k,l,m have the shape of the region1 of Fig. 2.5b. The *free walking area* of Fig. 2.11n has the shape of the *regions* of Fig. 2.5a, and of the region2 of Fig. 2.5b. Fig. 2.11m depicts a moment where a U-shape obstacle was partially visible. Thus, the *free walking area* is within the obstacle. However, structurally the *free walking area* still has the solution (the gap on the left-hand of the *free walking area*). Thus, the trap situation is avoided.

  Directions towards the obstacles were selected during almost all the experiment. Figs. 2.11c-k,e-l,f-m,g-n depict moments where the robot was directed towards an obstacle.

  In some parts of the experiment, motion directions far from the goal

direction were required. The Figs. 2.11m,n illustrate some of these moments.

The velocity profile is illustrated in Fig. 2.11b. The velocities are higher than in other experiments, because in many parts of the experiment the robot was in High Safety, see Fig. 2.7c. The time of the experiment was $83sec$, and the average translational velocity was $0.247\frac{m}{sec}$.

2.7 Comparison and Discussion

This Section presents a discussion regarding other collision avoidance approaches, the limitations of the ND method, and the limitations of the reactive approaches in general.

2.7.1 Discussion

The ND method avoids the **local trap situations** due to the environment structure (e.g. the U-shape obstacles, and moving among very close obstacles). The *free walking area* device is used to select a "navigable" region to move. When a U-shape obstacle is completely "visible", there are no *free walking areas* within the obstacle. Thus, the free space within the obstacle is not selected for motion, see Figs. 2.11k,l,n. Only, when the U-shape obstacle is partially visible, and there exist some symmetrical conditions involving the goal location, the robot might move towards the inside of the obstacle. Fig. 2.11m depicts an example of a partially visible U-shape obstacle that does not trap the robot. The ND method successfully navigates among very close obstacles, see Figs. 2.9d-k,f-m,g-n and Figs. 2.10e-o. Notice that, to select a *free walking area*, it is checked whether the robot fits among the obstacles.

The Potential Field Methods produce local trap situations due to the motion among close obstacles, and the U-shape obstacles [33]. Both create potential minima that trap the robot. To move a robot among close obstacles, the methods based on polar histograms [11], [59], [60] have the difficulty of tuning an empirical threshold. While one threshold is necessary to navigate among close obstacles, the threshold has to be modified to navigate in environments with no obstacle density. Traps due to the U-shape obstacles are not avoided by the methods that use constrained optimizations [25], [56], [4], [22]. The reason is that the optimization loses the information of the environment structure, which is necessary to solve these situations (the structure is studied with the *free walking area* in the ND method). There are methods based on a

given path that is deformed in real-time [54], [29], [14], [12]. When the path lies within U-shape obstacles dynamically created a trap situation appears.

The ND method computes **oscillation free motion** when the robot moves among very close obstacles. The Low Safety 2 action has been implemented to comply with this requirement. The motion is computed from the two closest obstacles. See the complete robot path and the velocity profile in Figs. 2.9a,b and Figs. 2.10a,b. The Potential Field Methods can produce oscillatory motion when moving among very close obstacles, or narrow corridors [33].

Motion directions far away from the goal direction are obtained with the ND method (**goal insensitivity**). The reason is that the goal direction is directly used in only one of the five motion laws (in High Safety Goal in Region, where the robot is not in danger, and there is not an apparent navigation complexity). This property was determinant in many situations encountered in the experiments, see Figs. 2.10h-q,i-r and Figs. 2.11e-l,g-n.

The reactive methods that make a physical analogy directly use the goal location in the motion heuristic: potential field methods [30], [34], [58], [29], [9], [50], the perfume analogy [6], the fluid analogy [43]. These methods exhibit high goal sensitivity. Thus, it is difficult to obtain directions far away from the goal location (in all the situations where they are required). The methods that solve the problem with a constrained optimization [25], [56], [4], [22] one of the balance terms is the goal heading. Therefore these methods also exhibit high goal sensitivity.

In the ND method action implementation, nothing prohibits the selection of **motion directions towards the obstacles**. Thus, directions towards obstacles are computed when required, see Figs. 2.9e-l,f-m,g-n, Figs. 2.10e-o,h-q,i-r, and Figs. 2.11k-c,l-e,m-f,n-g. Some methods explicitly prohibit the selection of motion towards the obstacles [59].

One difficulty found in most of the collision avoidance approaches is the **tuning of the internal parameters**. It is intricate to find the optimum values for a good behavior in all the collision avoidance situations. The ND method only has one parameter heuristically chosen (p parameter). This parameter is only used in one of the five navigation laws, it is a multiplier of a physical magnitude, and it is easy to find a value that does not determine the final method behavior.

The ND method uses five different situations and the associated actions to compute the motion commands. Hysteresis was required to smooth transitions between some situations.

Seen as a whole, the ND method is a robust reactive navigation method. This is mainly motivated by two facts:

1. The success of using a "divide and conquer" strategy to decompose the reactive navigation problem in sub-problems (by different situations). Specific strategies for motion are then developed for any situation.

2. The *free walking area* is a device that endows the method with the guaranty that: (1) it is possible to reach the goal, or (2) it is possible to reach the closest point to the goal, within the maximum reach of the local knowledge (sensory information).

2.7.2 Nearness Diagram Navigation Limitations

We think that the main limitation of the ND method is the method portability to different types of robots. The ND method does not take into account the so-called internal constraints of the robot: the shape, kinematic, and dynamic constraints. The ND method is designed for a circular holonomic robot working at low/medium velocities ($\simeq 0.5 \frac{m}{sec}$).

The methods of the related work take into account some of these constraints: the robot shape is considered in [30], [25], [4], and [22]. Some of them compute motion commands that comply with the robot kinematics: [59], [25], [4], [22], and [56]. The robot dynamic constraints are taken into account in [25], [4], [22], and [56].

To deal with non-circular shapes is a difficult problem for the ND method formulation. The ND method is formulated to apply over the Workspace, while the classical space used to represent the robot geometry is the Configuration space [39]. We have been working in this direction developing an under-constrained solution for square and rectangular shapes, see Chapter 3.

Taking into account the kinematic constraints in the ND method formulation, we have developed the *Ego-Kinematic Space*. By means of a simple transformation, the reactive navigation problem is carried out to a space where the robot is free of kinematic constraints. Standard reactive methods that do not take into account the robot kinematics (in particular the ND method) can be applied in this space, and as a consequence, the solution complies with the kinematic constraints, see Chapter 4.

We have been working to introduce the robot dynamic constraints in the ND method formulation. We have constructed a space where the dynamic constraints are implicitly represented - the *Ego-Dynamic Space*. Standard reactive methods are used in this space. As a consequence, the reactive method solution complies with the dynamic constraints. By using this research, the ND method velocity limits can be significantly increased, and still safety is guaranteed, see Chapter 5.

Another limitation of the reactive method design is that there is not a *Low Safety* situation whose action drives the robot towards the goal, when it is inside the *security zone*. This is a limitation of the reactive method design. However, the ND method does not have this limitation. The reason is that the robot can be in *Low Safety 2* created by a virtual valley, see Fig. 2.6. The robot would be driven towards the goal, irrespective of how close the goal is to the obstacle.

We have not addressed the sensor noise in the ND method implementation. We believe that external modules might process the sensory information in order to deal with noisy sensors, see [10]. However, strategies such as increasing the *security distance* according to the sensor uncertainty could be designed.

2.7.3 Improvements to all Reactive Approaches

The common limitation of all the reactive navigation methods analyzed in this Section, including the ND method, is that they are purely *local*: global convergence to the goal location cannot be guaranteed. Recently, some researches have worked on introducing *global* information into the reactive methods to avoid the global trap situations. [60] uses a look ahead verification to analyze the consequences of heading towards the candidate directions. The method avoids the trap situations by running the algorithm a few steps in advance of the algorithm execution. [14], [12] and Chapter 6 exploit the information of the connectivity of the space using a navigation function. This provides the reactive method with global information used to avoid the trap situations. Moreover, these two approaches take into account the so-called external constraints: the nature of the environment. Both methods are adapted to work in highly dynamic scenarios.

2.7.4 Nearness Diagram Navigation Background

The ND method has been validated on the *Diligent*[5]robot at LAAS-CNRS (France). The reactive method has been successfully integrated as the low-level motion generator in the $G^{en}oM$ architecture [23], and the method is daily used for demonstrations, see [1]. In the *Robels* system [51], the ND method is one of the five sensory-motor functions used to move the robot. Two other sensory-motor functions are evolutions of the ND method, see Chapter 6.

With some modifications, the method is now working in other indoor/outdoor

[5]Diligent is a Nomadic XR4000 platform.

mobile platforms: *Hilare*, *Hilare 2*[6], and *Lama*[7] at LAAS-CNRS (France); *Otilio*[8] at University of Zaragoza (Spain), and *r2*[9] at Technical University of Lisbon (Portugal), see Chapter 3. Currently, the method is being implemented on *Dalay*[10] at LAAS-CNRS (France).

2.8 Conclusions

This Chapter presents the design of a reactive navigation method. It has been designed using the *situated-activity paradigm* of behavioral design. The design of the method is the basis to implement reactive navigation methods. The advantage is that the paradigm used in the design employs a "divide and conquer" strategy to reduce the difficulty of the problem. As a consequence, the reactive navigation methods implemented, following the presented guidelines, might be able to successfully navigate in more troublesome scenarios than other methods do.

At the current moment, the method design has been used to implement some reactive navigation methods that adapt to their collision avoidance context. For example the *Free Zone Method* [41], [42] for soccer player robots. We have used the method design to implement the ND method. The main contribution of this method is that it robustly achieves navigation in very dense, cluttered, and complex scenarios. These environments are a challenge for many existing methods.

The method can be extended to three dimensions. The information collected to define the situations, and the geometry-based implementation of the actions, both can easily be redefined to the third dimension. This will allow for generating reactive collision avoidance for free-flying objects working in three-dimensional workspaces.

[6]Hilare and Hilare2 are indoor rectangular differential-driven robots. The main sensors used were two 2D planar laser.

[7]Lama is a rectangular outdoor robot that can work in differential-driven mode. The sensor used was a pair B/W cameras.

[8]Otilio is square and differential-driven indoor robot. The sensor used was a 3D laser.

[9]r2 is a circular and differential-driven indoor robot. The sensor used was a ring of ultrasounds sensors.

[10]Dalay is a rectangular and differential-driven outdoor robot. The sensor being used is a 2D laser.

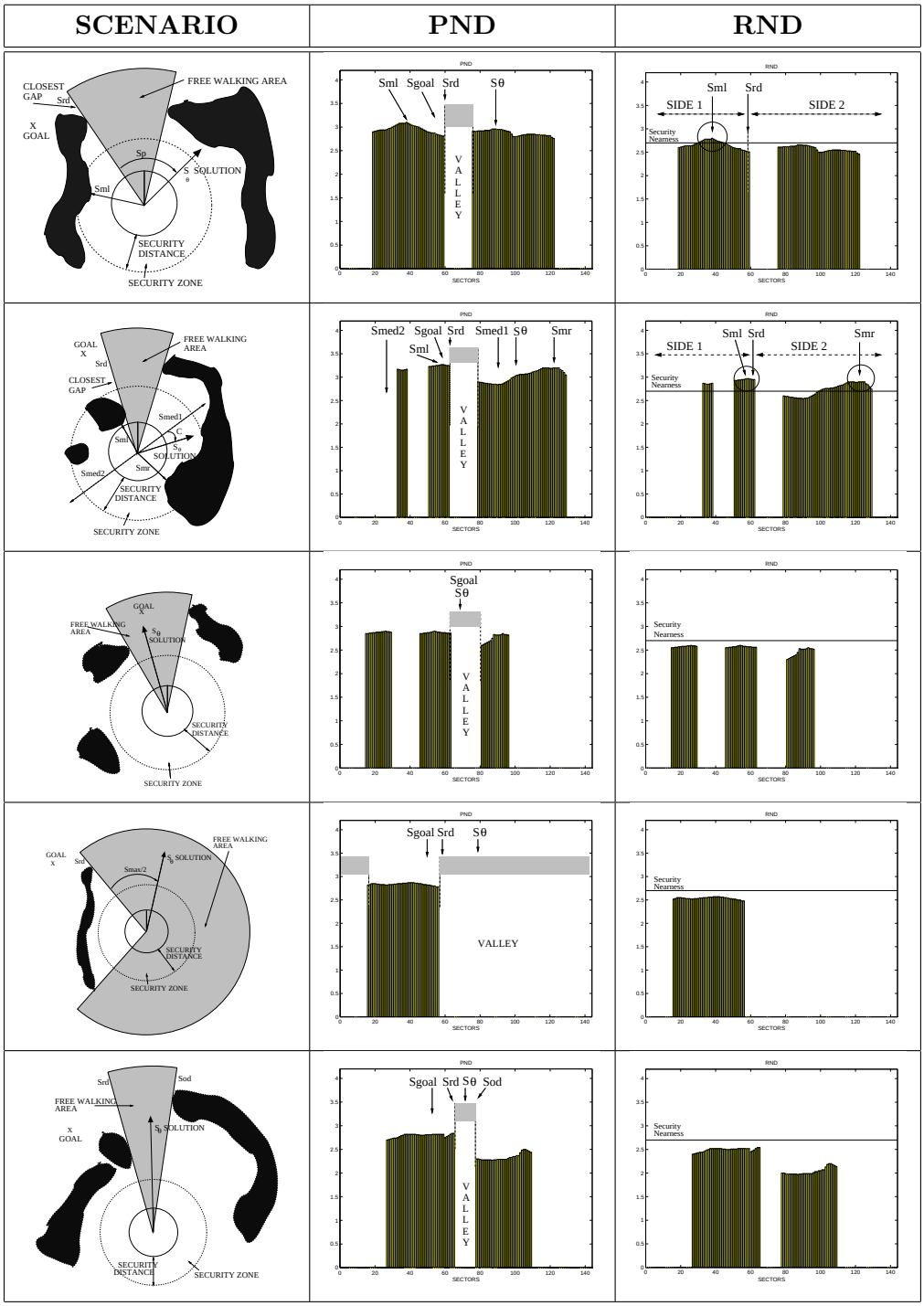
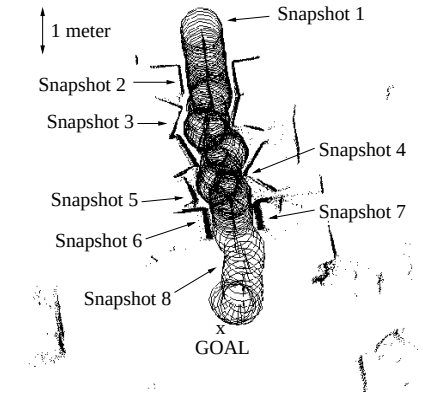
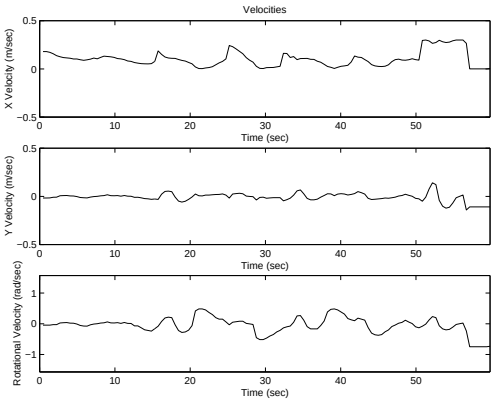


Figure 2.8: Situation/Action table and the PND and RND diagrams. Row 1: LS1. Row 2: LS2. Row 3: HSGR. Row 4: HSWR. Row 5: HSNR.



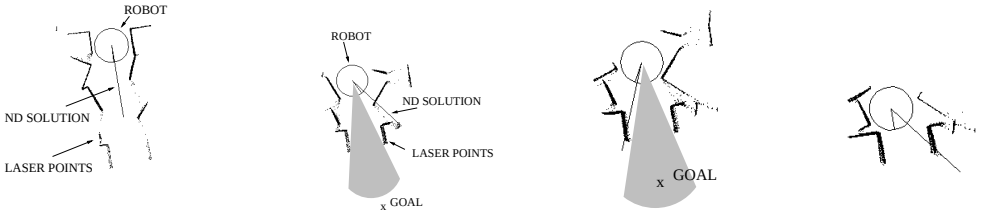
(a)



(b)

(c) (d) (e) (f)

(g) (h) (i) (j)



(k) (l) (m) (n)

Figure 2.9: Experiment 1.

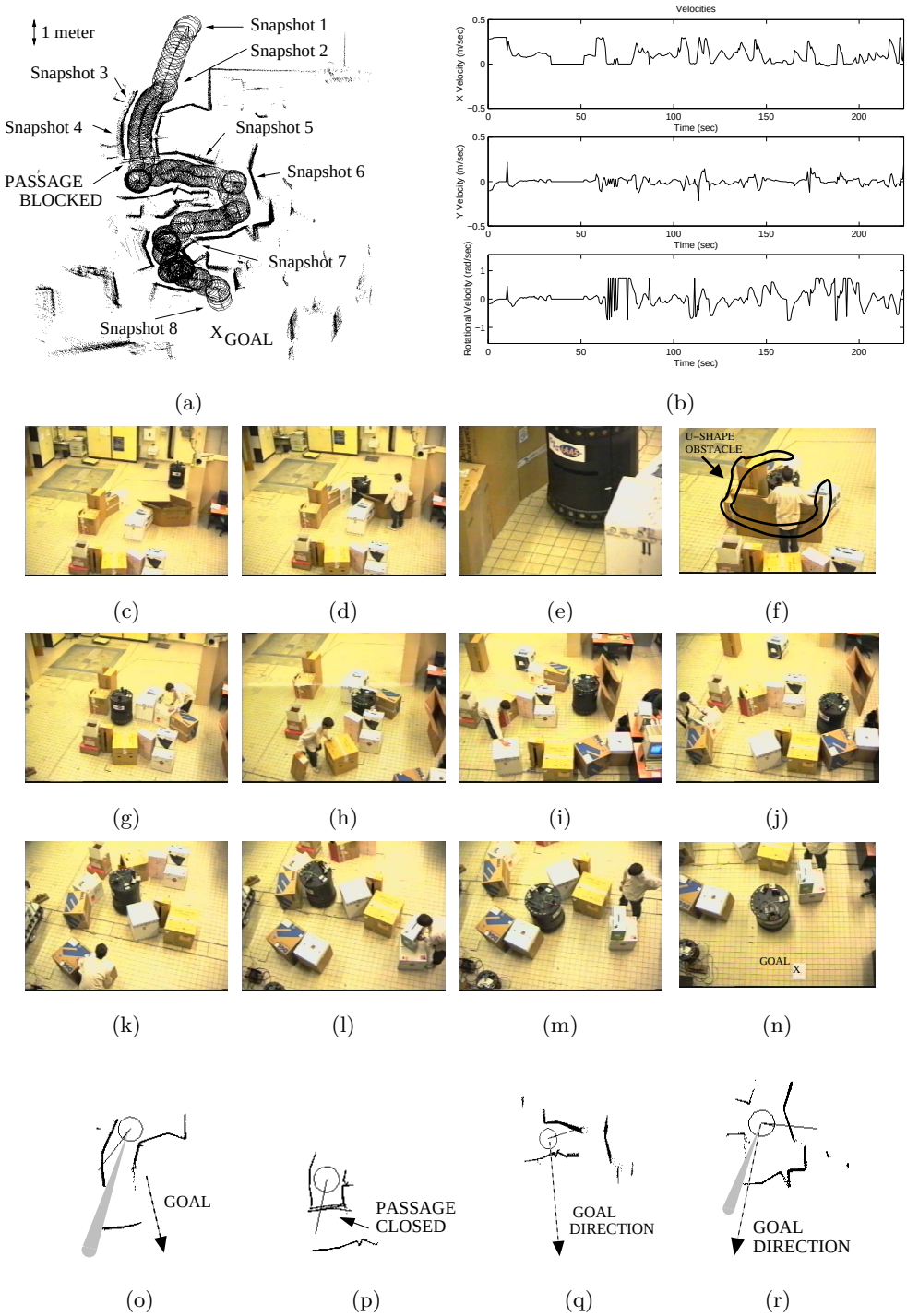


Figure 2.10: Experiment 2.

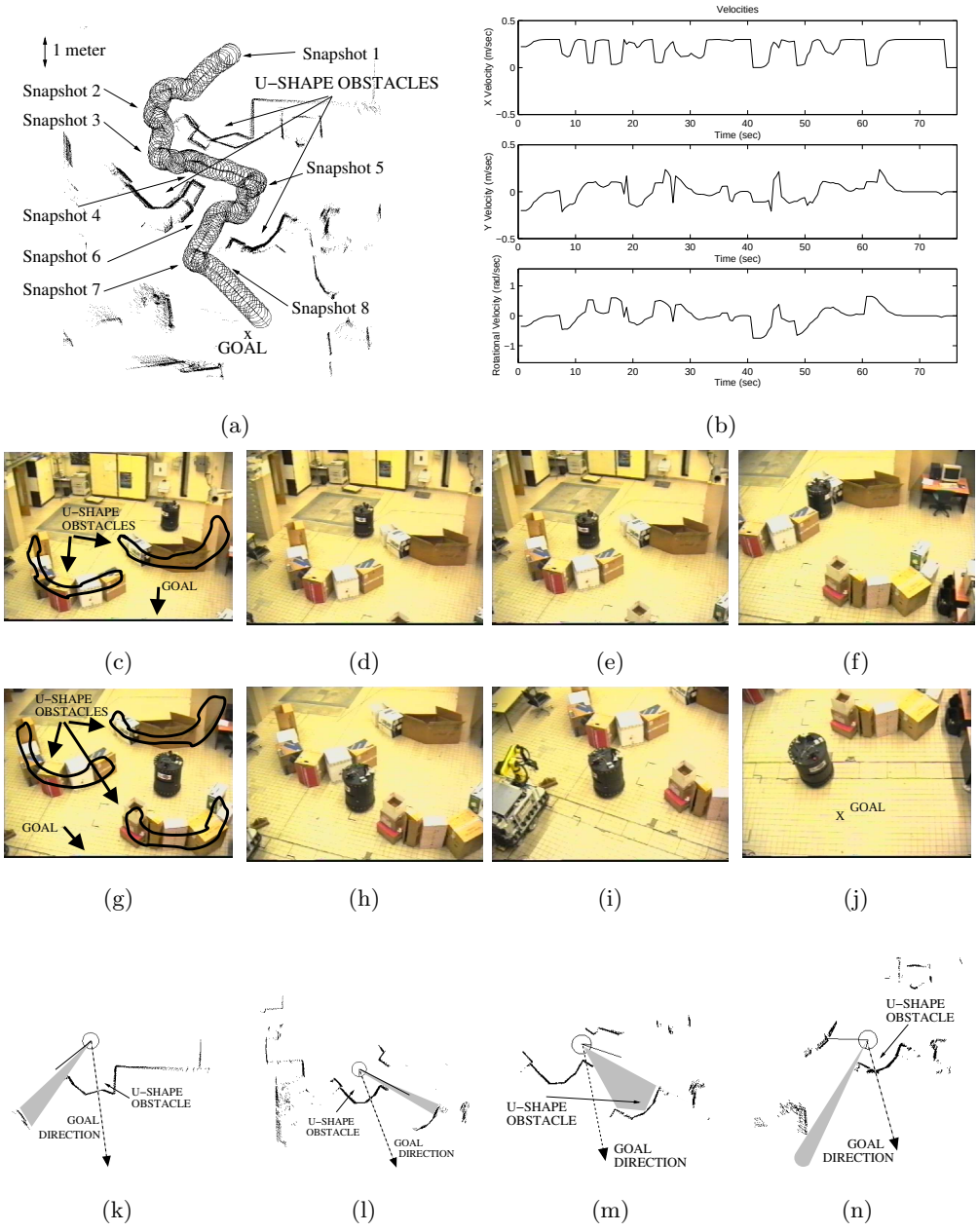


Figure 2.11: Experiment 3.

Chapter 3

Under-constrained Sensor-Based Motion Planning

3.1 Introduction

In typical mobile robotic missions, the indoor/outdoor environment is unknown and non-predictable (e.g. offices, museums, planetary surfaces). From the navigation point of view, the different nature of these scenarios imposes a wide range of difficulties: they are unstructured, dense, cluttered, etc. In addition, the robots designed usually exhibit different shapes, kinematics, and dynamics. In these scenarios, the robot shape, kinematics, and dynamics play an important role, since they constrain the possible robot motions.

This Chapter presents an under-constrained solution to address the shape, kinematics, and dynamics in the reactive navigation layer. This solution is used to extend the *Nearness Diagram Navigation* (Chapter 2) to compute commands that comply with these constraints.

The solution proposed is under-constrained since the shape, kinematics, and dynamics are taken into account after the reactive method usage. Thus, while the motion commands comply with the constraints, they are not used from the beginning in the motion commands computation.

The advantage of the framework is the integration methodology, the easy portability among platforms, and the navigation results. The method successfully achieves navigation, with different robots and sensors, in very dense, complex, and cluttered scenarios. To demonstrate the easy portability among

different platforms, the method has been tested on four indoor robots and on one outdoor robot at three different laboratories.

This research was previously presented in [45].

The paper is distributed as follows. Section 3.2 discusses related work. Section 3.3 introduces the framework and Section 3.4 presents the experimental results. Finally, in Section 3.5 the conclusions are drawn.

3.2 Related Work

To design the safe motion generation task, there are at least three issues to address: the navigation method, the robot kinematics and dynamics, and the robot shape.

The **navigation technique** used is a *reactive navigation method* [30], [11], [59], [25], [44], [54], among others. Based on a perception-action process, these methods compute collision-free motion commands in a goal-directed fashion. The main advantage is that these methods require a low computational load. This provides the system with the possibility of a high-rate environmental feedback. The drawbacks of these approaches are that they produce sub-optimal solutions, and they cannot guarantee the reach of the goal location. A deeper discussion about these methods is presented in Chapter 2.

Currently, most of the robots designed exhibit **kinematic** and **dynamic constraints**. For these robots, the navigation method has to take into account these constraints. Otherwise, the robot cannot execute the motion computed by the reactive navigation method [30], [11], [44], [54]. This issue will be discussed in detail in Chapters 4, and 5.

The robot **shape** is a difficult problem in reactive navigation. Classically, the robot shape is taken into account by carrying out the problem to the Configuration space [35]. In this space, a point represents the robot. To map the obstacles into the Configuration space is a time-consuming task not complying with the real-time requirement. Some reactive navigation methods avoid the Configuration space computation, by approximating the robot by circular shapes. Thus, the collision checking can be carried out in the Workspace [11], [44], [54]. If these methods are used over non-circular bases, the robot shape needs to be approximated, under-constraining the method solution. This issue will be discussed in detail in Chapter 4.

This Chapter addresses the robot motion in very dense, complex, and cluttered scenarios. The robot shape, kinematics, and dynamics are determinant in these scenarios, and thus, they cannot be neglected. The *Nearness Diagram Navigation* method is a reactive navigation method that successfully navigates

in those scenarios. However, the *Nearness Diagram Navigation* method does not consider the robot shape, kinematics, and dynamics. For this reason, this Chapter addresses the extension of the *Nearness Diagram Navigation* method to take into account the robot shape, kinematics, and dynamics.

3.3 The Framework

The goal of this Section is to understand the solution proposed to extend the *Nearness Diagram Navigation* method to address the shape, kinematics, and dynamics.

The solution is based on breaking down the motion computation process into three sub-problems related to: (1) collision avoidance, (2) the robot kinematic and dynamic constraints, and (3) the robot shape:

1. The reactive collision avoidance approach - *Nearness Diagram Navigation* - is used to calculate the most promising motion direction and the desired velocity. The assumption is a circular robot free of motion constraints.

2. The *Motion Generator* uses the information provided by the *Nearness Diagram Navigation* method to compute a motion command that complies with the kinematic and dynamic constraints.

3. For non-circular robots, the *Shape Corrector* modifies the pre-computed motion command to take into account the robot shape.

Fig. 3.1 illustrates the complete process. Seen as a whole, this framework computes collision-free motion commands to drive a mobile platform towards a given goal location. The motion commands comply with the robot kinematics and dynamics, and the robot shape is considered.

The three modules are next analyzed.

3.3.1 Nearness Diagram Navigation

The *Nearness Diagram Navigation* (ND) method is described in detail in Chapter 2. However, a brief description of the method is next presented. The ND method is based on the situated-activity methodology of design [3]. First, a set of five situations that fully describe the relative state of the robot, obstacle distribution, and goal location are defined. Subsequently, one action is designed for each situation. In real-time, the sensory information is used to

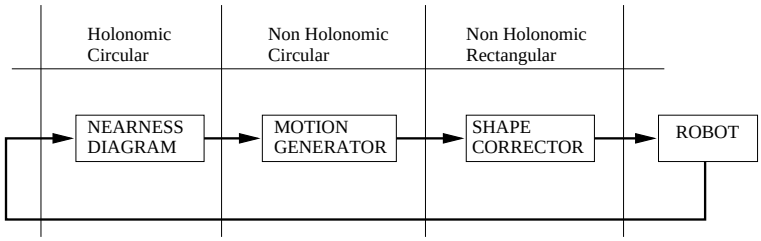


Figure 3.1: The reactive navigation problem broken down into subproblems.

identify the current situation, and the associated action is executed computing the motion commands (translational velocity v_{ND}, and velocity direction θ_{ND}).

Good results in very cluttered, complex, and dense scenarios have been reported using the ND. This is motivation to select the ND for the framework. However, the ND does not take the kinematic and dynamic constraints, and non-circular shapes into account.

3.3.2 The Kinematic and Dynamic Constraints

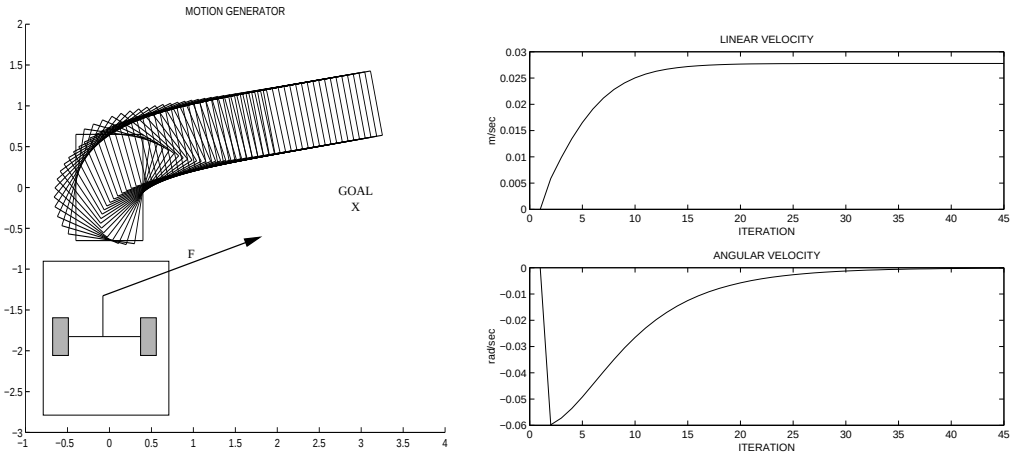
The ND computes the most promising motion direction, θ_{ND}, and the desired velocity, v_{ND}. Both motion commands still need to be converted into motion commands for the non-holonomic mobile base. This issue is achieved by the *Motion Generator*, see Fig 3.1.

The *Motion Generator* (MG) [5] is a dynamic model-based robot controller for differential-drive robots. The controller has physical parameters: inertia, friction, application point of the force. These parameters are for tunning the controller behavior, thus, there is no need to identify them. The MG computes the velocity commands from a virtual force applied to a point on the robot at each sample period. The MG has the following advantages:

1. The model takes into account the kinematic and dynamic constraints. Thus, the controller computes feasible commands for the real robot.

2. The model parameters have a clear physical sense. This allows an easy tuning of the parameters in order to obtain the desired dynamic behavior (non overshooting during turns, and limits in velocities and accelerations).

3. The model filters the sudden changes in direction produced by many reactive navigation methods.



(a) (b)

Figure 3.2: a) Robot trajectory. b) Translational and rotational velocities (motion command) obtained with the MG.

Fig. 3.2a depicts the robot moving between two locations. The input of the MG is a force, $\overrightarrow{\mathbf{F}}$, calculated at each sampling period. The force "pulls" the model towards the goal location. The output of the MG is the motion command given to the differential-driven robot, $\mathbf{v} = (v, w)$. See Fig. 3.2b.

The *Nearness Diagram Navigation* method and the *Motion Generator* are connected as follows. The most promising motion direction, θ_{ND}, and the velocity, v_{ND}, obtained with the ND are transformed into a force, $\overrightarrow{\mathbf{F}} = (|F|, \theta_F)$, the input of the MG:

$$|F| = F_{max}.\frac{v_{ND}}{v_{max}} \ , \quad \theta_F = \theta_{ND} \tag{3.1}$$

This framework computes the collision-free motion commands to drive a circular and differential-driven robot towards the goal location.

3.3.3 The Shape Constraint

The *Nearness Diagram Navigation* method and the *Motion Generator* compute collision-free motion commands for a circular and differential-driven platform. The reactive navigation method only takes into account circular shapes. Thus, other robot shapes are not considered. The *Shape Corrector* is used to extend this framework to take into account other shapes, see Fig. 3.1. From now on,

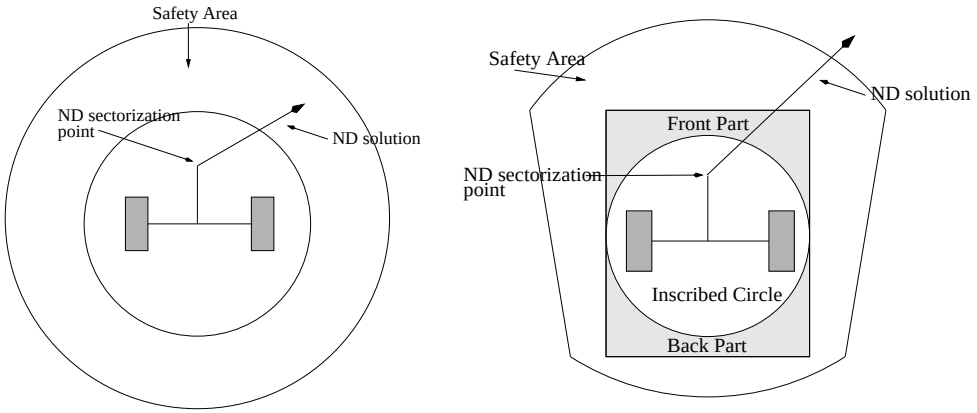


Figure 3.3: a) Circular robot. b) Rectangular robot.

the attention is focused on rectangular shapes (the square shape is a particular case).

In a first step, the - ND + MG - framework is used to compute a motion command approximating the robot shape by the inscribed circle, see Fig. 3.3b. (Very dense, complex, and cluttered scenarios motivate this non-conservative selection: the circumscribed circle is a coarse approximation). The *Shape Corrector* modifies the pre-computed motion command, to protect from collisions the two parts of the robot outside of the inscribed circle (the front and back parts), see Fig. 3.3b.

The - ND + MG - computes motion commands that produce instantaneous forward motion $v \geq 0$. (The reactive method is constrained to compute instantaneous forward directions of motion) Then, the *Shape corrector* is based on three situations that exploit this constraint:

1. **Imminent collision**: The robot is in this situation when the collision cannot be avoided with any sequence of forward motions. The motion command stops the robot ($v = 0, w = 0$), because the collision is imminent. A flag is then fired to launch a higher-level module to bring the robot out of this situation (e.g. a motion planner). Fig. 3.4a illustrates this situation: when there are obstacles inside the *Emergency Stop Area* (calculated from the circumscribed circle to the robot shape), no motion with $v \geq 0$ (forward motion) or sequence of them avoid the collision.

2. **Front collision danger**: The robot is in this situation when there is a

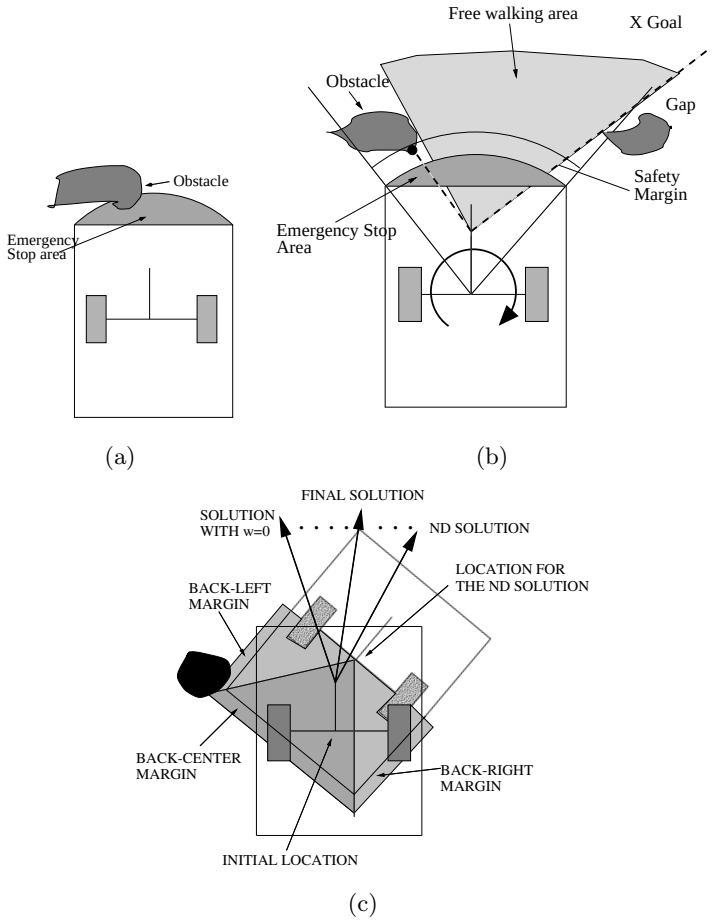


Figure 3.4: a) Inminent collision.. b) Front collision danger. c) Back collision danger.

potential risk of entering into an *Imminent collision* situation. A *Safety Margin* is defined to enclose the *Emergency Stop Area*, see Fig. 3.4b. The robot is in this situation when there are obstacles within the *Safety Margin*. The motion command stops and rotates the robot over its center ($v = 0$,w) to clear the *Safety Margin* of obstacles. The selection of the rotation direction depends on: (1) the closest obstacle inside the *Safety Margin*, and (2) an internal piece of information of the *Nearness Diagram Navigation* method called - *free walking area* (for wider details see Chapter 2). The rotation direction is computed as follows:

(a) If the *free walking area* gap closest to the goal is on the right-hand

of the direction of the closest obstacle inside the *Safety Margin*, then the robot rotates towards the right. If not,

(b) The robot rotates towards the left.

The result of this strategy clears the *Safety Margin* of obstacles, implicitly avoiding the *Imminent collision* situation. Fig. 3.4b illustrates an example of this situation, in which the robot stops and rotates towards the right.

3. **Back collision danger**: The robot is in this situation when the pre-computed motion command produces a collision with the back part of the robot. To detect this situation, the trajectory that would be obtained under the execution of the pre-computed motion command is analytically calculated (following [25]). Then, the robot motion is simulated over the trajectory checking collisions, to validate the motion command. The back part of the robot is enlarged for safety reasons. The back of the robot is divided into three parts *Back-Left, Back-Right and Back-Center Safety Margins*, see Fig. 3.4c. Then, three cases distinguished:

- A collision is detected with the *Back-Left Margin*. The angle between the ND solution and the direction that would produce $w = 0$, is discretized in sub-directions. The MG computes a new motion command by using one of those sub-directions. The new motion command is tested for collisions over the trajectory. This procedure is repeated until a collision-free motion command is found. Fig. 3.4c illustrates an example where the pre-computed motion command produces a collision with the *Back-Left Margin*. Some intermediate directions are tested, until a collision-free command is found.

- A collision is detected with the *Back-Right Margin*. The above procedure is repeated but towards the right-hand direction.

- A collision is detected with *Back-Center Margin*, or simultaneously with *Back-Left and Back-Right Margins*. The motion command is a forward motion without rotation ($v \geq 0$,$w = 0$).

This framework computes the collision-free motion commands to drive a rectangular and differential-driven robot towards the goal location.

3.4 Experimental Results

This Section validates experimentally the proposed research. This framework has been tested on four indoor robots and on one outdoor robot. The main characteristics of the robots are: (1) they are differential-driven robots, with the exception of the outdoor robot that can be set to work in differential-driven mode. (2) The robots have circular, square, and rectangular shapes. (3) The on-board sensors are ultrasounds, 2D and 3D laser rangefinders, and a stereo pair of cameras.

To adapt the framework to each robot, the main adjustments were: (1) an special safety area (for each robot shape) was designed for the *Nearness Diagram Navigation* method, see Fig. 3.3a and Fig. 3.3b. (2) The *Motion Generator* parameters were tuned to have the desired dynamic behavior compatible with each platform constraints (due to the different motion capabilities of each robot). (3) Only some geometric parameters were changed in the *Shape Corrector*.

In all the experiments the environment was completely unknown, and only the goal location was given in advance to the robot. The environments were unstructured (random shaped obstacles). Nevertheless, the environment could be dynamic and non-predictable. Under all these circumstances it is widely justified the use of reactive navigation algorithms to move the robot.

The experiments reported have a common objective: To show that this framework is able to safely drive a kinematic and dynamic constrained mobile platform in very dense, complex and cluttered scenarios. This is the type of scenarios where other approaches are susceptible to failure for the following reasons: (1) the reactive method itself, (2) approximations in the motion executed, and (3) approximations in the robot shape.

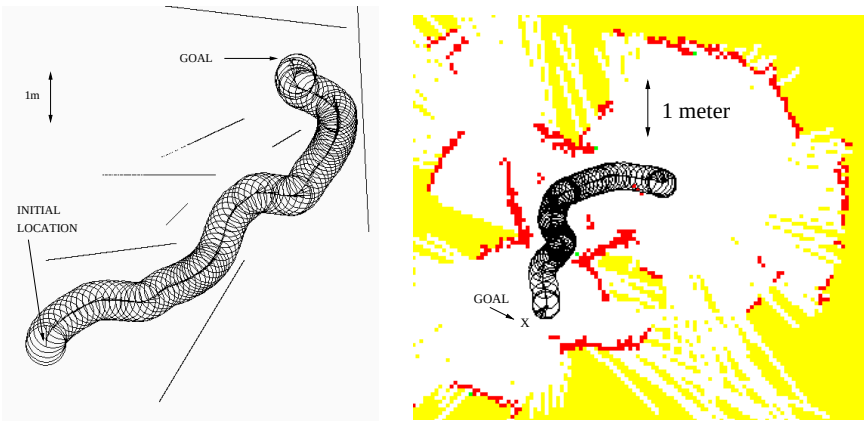
3.4.1 Circular robots

For circular robots the *Nearness Diagram Navigation + Motion Generator* framework is used.

Nomadic Scout

This framework has been implemented in a *Nomadic Scout* at the Instituto Superior Técnico de Lisboa, Portugal, see Appendix E.0.7.

Fig. 3.3a illustrates the robot model and the point used to apply the ND. The sampling time was $T = 0.25sec$ and the maximum speed $v = 0.3\frac{m}{sec}$.



(a) (b)

Figure 3.5: a) Simulated experiments with laser. c) Real experiment with ultrasounds.

The framework was tested on a simulator and on the real robot. On the simulator a simulated SICK laser were used. On the real robot a ring of ultrasounds was used. The ultrasound measurements were processed to build depth maps [10]. The experiments showed good results in dense, complex and cluttered scenarios. See Fig. 3.5a,b.

3.4.2 Rectangular and square robots

For square and rectangular robots the *Nearness Diagram Navigation + Motion Generator + Shape Corrector* framework is used.

Labmate platform

This framework has been implemented and tested on a *Labmate* platform at the Universidad de Zaragoza (Spain), see Appendix E.0.5.

For the experiments, the 3D *TRC* laser was used. The last 20 laser measurements, projected onto the floor and corrected to the actual robot location, were used as a short-time memory. The sampling period was set to $T = 0.4sec$ and the maximum speed to $0.3\frac{m}{sec}$.

Fig. 3.6. shows an experiment in a typical indoor environment. The robot passed through two doors (the last one was half-open) and a corridor full of obstacles. The goal location was successfully reached without any collisions with the environment.

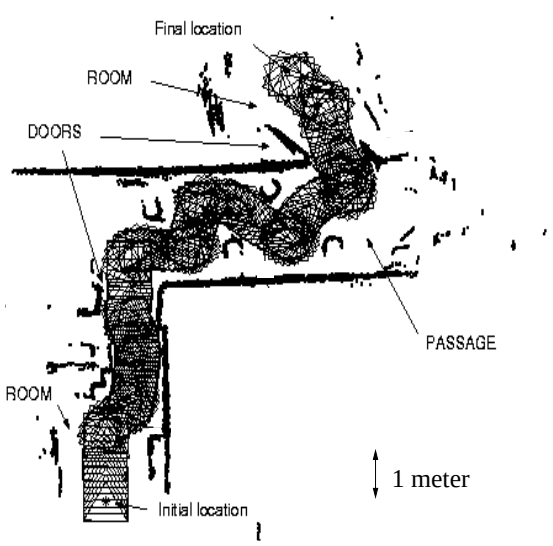


Figure 3.6: Real Experiment the *Labmate*.

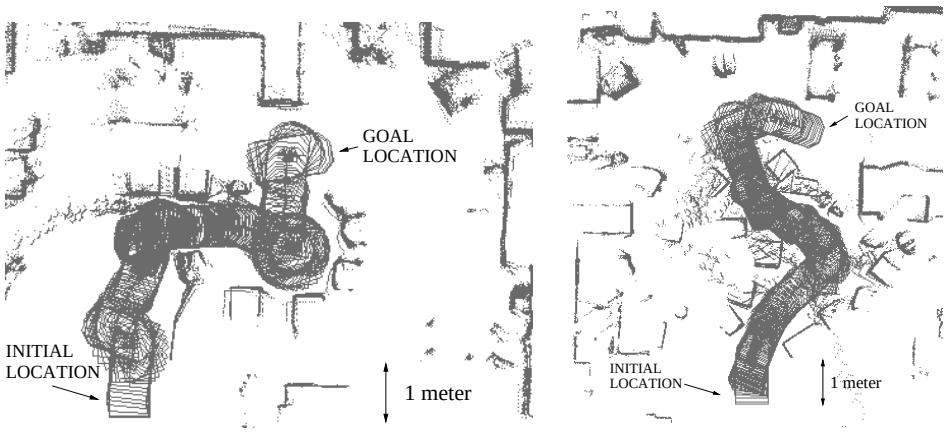
Hilare2 and Hilare2Bis

This framework has been implemented and tested on the *Hilare2* and *Hilare2Bis* platforms at LAAS (CNRS), France, see Appendix E.0.6.

For the experiments, a SICK was used. The last 40 laser measurements, corrected with the robot odometry, were used as a short-time memory. The sampling period was set to $T = 0.4sec$ and the maximum speed to $0.3\frac{m}{sec}$. To take the inertia into account on *Hilare2Bis*, some parameters of the *Motion Generator* were tuned to have the desired dynamic response.

Fig. 3.7a shows an experiment with *Hilare2* in a very dense, complex and cluttered scenario. The robot navigated along a very narrow corridor ($< 10cm$ on both sides). Next the robot maneuvered reactively in a constrained space (central part of the experiment) to turn towards the exit. The goal location was successfully reached without collisions with the environment.

Fig. 3.7b shows an experiment with *Hilare2Bis* in a troublesome scenario built while the robot was moving (this creates the environment dynamic component). The robot navigated along an asymmetric corridor, and in the last part of the experiment, the robot maneuvered to the right side to reach the goal location. The robot successfully arrived to the goal location without



(a) (b)

Figure 3.7: a) Real Experiment with *Hilare2*. b) Real Experiment with *Hilare2bis*.

collisions with the environment.

Lama

This framework has been implemented and tested on the outdoor *Lama* platform at LAAS-CNRS, France, see Appendix E.0.8.

The obstacle information is computed with a probabilistic obstacle detection procedure [27]. The perceived area is described by a set of polygonal cells. By means of a Bayesian classifier, the cells are labeled with the probability that an obstacle occupies them. The image-processing period is about four seconds. The sampling time was set to $T = 0.4sec$ and the maximum speed $10\frac{cm}{sec}$.

The experiment was conducted in a typical outdoor environment, see Fig. 3.8a. The framework drove the robot towards the goal location avoiding collisions with the obstacles. The complete run was about $20m$.

3.5 Conclusions

The presented framework generalizes a reactive navigation method - *Nearness Diagram Navigation* - to work on robots with kinematic and dynamic constraints. This framework computes reactively motion commands to drive a mobile platform towards a goal location, whilst avoiding collisions with the environment.

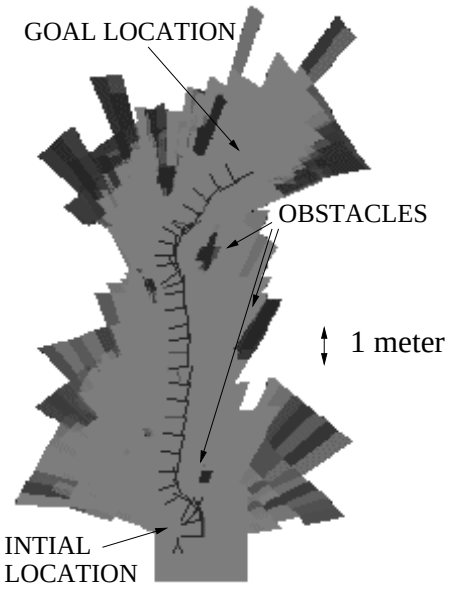


Figure 3.8: Real Experiment with *Lama*.

Experiments in very dense, complex and cluttered scenarios have been reported, where: (1) it necessary the use of a reactive navigation method able to deal with these troublesome scenarios. (2) The robot kinematic and dynamic constraints have to be taken into account, because they limit the motion capabilities. (3) The robot shape also reduces the possible collision-free motions, and it has to be taken into account.

To validate the proposed approach, and to demonstrate the easy portability among different platforms, the framework has been tested on five different robots at three different laboratories. The results were very satisfactory, and the platforms were safely moved during hours of tests.

From the author's point of view, the limitation of the framework is the decomposition of the navigation problem in sub-problems. This is an under-constrained solution, since the kinematics, dynamics, and shape are not explicitly taken into account in the reactive method stage. However, the navigation results are outstanding because the typical mobile platforms have quick dynamics, and the kinematics are not very significant with the small sample periods used. On the other hand the result would be degraded. This issue is analyzed in detail in Chapters 4 and 5.

Chapter 4

Sensor-Based Motion Planning with Kinematic Constraints

4.1 Introduction

Even though many robots exhibit kinematic constraints, most sensor-based navigation methods do not take these constraints into account. This is a hard limitation when using a navigation method. The robot can only approximately execute the motion computed: safety cannot be longer guaranteed. In other cases, some sensor-based algorithms have been extended to address the motion constraints. However, with this approach, the reactive navigation method is usually re-designed from scratch to address the constraints. And a new re-design might be necessary to incorporate the constraints of another robot.

There is a lot of literature that proposes particular solutions to address the motion constraints in the sensor-based motion planning field. But little attention has been paid to look for much wider solutions.

This Chapter presents a novel idea to take into account the kinematic constraints: to introduce them in the space. The proposal is to use the kinematics to construct a novel spatial representation, the *Ego-Kinematic Space*. The kinematic constraints are embedded in the space, so the robot in this space moves as a "free-flying" object. Then, standard sensor-based algorithms that do not take into account the kinematic constraints can be used in this space. As a consequence, the algorithms comply with the kinematics without addressing it directly. By using this approach, it is straightforward to apply existing

reactive navigation algorithms to non-holonomic robots. An ample solution, to address the motion constraints in the sensor-based motion planning discipline, is presented.

Another problem overcame in this Chapter is the space dimensionality. The Configuration space for non-holonomic robots such as differential-driven robots, tri-cycle robots, and car-like robots is three-dimensional $\mathbb{R}^2 \times S^1$. In the sensor-based motion planning context, the computations required to deal with three dimensions are a limitation: these algorithms have to work in real-time. The necessity of using only a subspace of the Configuration space with the same structure of the Workspace, $\mathbb{R}^2$, is demonstrated. As such, the dimension of the Configuration space is reduced by one within the context of sensor-based motion planning. This will alleviate significantly the computation time of future sensor-based algorithms that make use of this result. In particular, the research presented in this Chapter.

The main advantage of this framework is that it can be utilized by related work. Many sensor-based algorithms, that do not take into account the kinematic constraints, can be used in the proposed spatial representation, without further concern about the kinematic constraints. Experimental results involving a non-holonomic robot are shown to validate the framework. Two methods were used for navigation (*Nearness Diagram Navigation*, Chapter 2, and the *Potential Field Method* [30]). They both originally do not address kinematics. By using the framework both methods were successfully used to safely drive the constrained platform among locations.

This research was previously presented in [47].

The Chapter is organized as follows: In Section 4.2 a general background for the rest of the Chapter is introduced. Section 4.3 discusses the Configuration space dimensionality. The *Ego-Kinematic Space* is addressed in Section 4.4. Finally, Section 4.5 shows the experimental results and Section 4.6 draws the conclusions.

4.2 Background and Preliminaries

This Section reviews some concepts used in the remaining part of the Chapter. Firstly, the robot kinematics covered by the analysis is presented. Secondly, the sensor-based motion planning techniques are discussed. Finally, the admissible paths that arise from the robot kinematic model are examined.

4.2.1 Robot Kinematics

Robots moving on a flat surface with the classical hypothesis "rolling without slipping" are addressed. The Workspace, $\mathcal{W}$, and the Configuration space, $\mathcal{C}$, are $\mathbb{R}^2$ and $\mathbb{R}^2 \times S^1$ respectively. A robot configuration is represented by its location and the orientation $\mathbf{q} = (x, y, \theta)$. The attention is focused on robots whose motion is constrained by:

$$-\dot{x}sin\theta + \dot{y}cos\theta = 0 \tag{4.1}$$

Equation 4.1 is a **non-holonomic equality constraint**. The effect is to reduce by one the dimension of the space of differential motions at any given configuration [7]. Hence, a motion command of the robot can be described by two motion parameters only.

The kinematic model of the *two-driving wheels* and the *car-like* mobile robots is next described. These robots are representative examples of mobile platforms that verify the constraint of Equation (4.1), see Fig. 4.1. A deeper description of both models is found in [36].

The kinematic model of both, the two-driving wheels and car-like robots, can be expressed by the following equation (the difference is a change of variable[1][2]), as detailed in [36]:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} cos\theta \\ sin\theta \\ 0 \end{pmatrix} v + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} w \tag{4.2}$$

where v and w denote the linear and angular velocities.

The main difference between these two platforms is that the car-like robot has a mechanical constraint. This imposes a maximum curvature γ_{max} (or minimum turning radius $r_{min} = \gamma_{max}^{-1}$) of the path executed by the robot. There are other robots (e.g. tri-cycle robots) whose kinematic models can also be expressed by Equation (4.2), and verify Equation (4.1).

4.2.2 Sensor-Based Motion Planning

To address the goals of this Chapter the role of the sensor-based navigation methods in generating collision-free motion is discussed.

[1] For the differential-driven robot, the controls are speeds of the wheels, v_r and v_l. The variable change is $v = \frac{1}{2}.(v_r + v_l)$ and $w = \frac{1}{l}.(v_r - v_l)$.

[2] For the car-like robot, the controls are the steering angle and the linear speed, ϕ and w'. The variable change is $v = L.w'.cos(\phi)$ and $w = L.w'.sin(\phi)$.

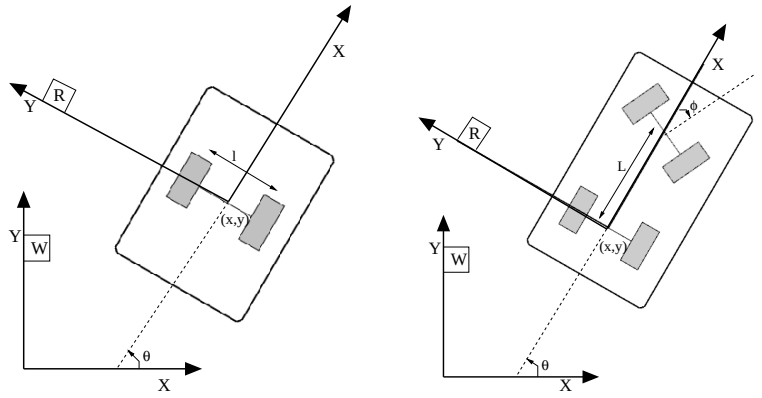


Figure 4.1: a) Two-driving wheels robot. b) Car-like robot.

The reactive navigation methods are based on a perception-action process. They compute collision-free motion commands, that drive the robot towards the goal. The result of applying repeatedly a reactive navigation method is a sequence of motion commands that move the robot from the initial location towards the final location, while avoiding collisions. Examples of these methods include [30], [58], [29], [9], [50], [22], [11], [28], [56], [25], [54], Chapter 2. These on-line methods are commonly accepted to move a robot in dynamic, unknown, and unstructured environments.

The vehicle kinematic constraints are still a subject of research in sensor-based motion planning. However, some methods have been proposed that deal with the kinematics. Mainly they parameterize a set of admissible trajectories to select the better later on [59], [60], [62], [28], [22]. Others carry out and solve the problem in the admissible motion command space [25], [56], [52]. Or they use a metric induced from the kinematic constraints [54].

This research is based on a spatial representation prior to the reactive method usage. The following analysis covers the spaces where in general the reactive navigation methods apply: the Workspace (e.g. [11], [25], [22], [56], [14]) and the Configuration space (e.g. [30], [59], [53]).

4.2.3 Admissible paths

The kinematic admissible paths have been mainly studied in motion planning from the shortest path point of view. The optimal paths between configurations are a concatenation of arcs of circle and straight segments for both, a car

that can only move forward [21], or that can also move backwards [55]. The idea of using the length of a kinematic admissible path as a distance is related to this research. The shortest path complying with the kinematic constraints is used as a distance for a non-holonomic metric in [37], and [49]. Other distances are computed from a robot configuration to an obstacle point [8], or to a segment [61], both in $\mathbb{R}^2$.

The interest in this Chapter is focused on the sensor-based motion planning context. The reactive navigation methods compute at each time a *motion command*. Under the execution of a single motion command, the admissible paths are **arcs of circle or the straight segment** (when dynamics are ignored, see [25] for an upper bound of the approximation error). This characterization of the paths has been widely used in sensor-based motion planning to address the kinematic constraints, see [59], [60], [62], [28], [22], [25], [52], among others.

4.3 Properties of the Workspace and the Configuration-Space

The approach described in this Chapter lies in applying a spatial transformation, prior to the application of a reactive navigation method, in such a way that the robot kinematic constraints are directly represented. This transformation must be applied to the robot Workspace or Configuration space, depending on the navigation method used.

The goal of this Section is to show that the Workspace and the subspace of the Configuration space needed, are both represented by $\mathbb{R}^2$ (within the contest of sensor-based motion planning). Moreover, the obstacle information in both spaces is also represented by $\mathbb{R}^2$.

As the Workspace is readily defined as $\mathbb{R}^2$, the attention is focused on the Configuration space, $\mathbb{R}^2 \times S^1$ (S^1 is the unit circle). In the context of reactive navigationm, those paths in the Configuration space obtained under the execution of a single motion command are considered. The set of all the robot configurations reachable by such paths is a subset of the Configuration space, the *Reachable Set of a single Motion Command*, denoted by $\mathcal{R}^{1mc}$, where $\mathcal{R}^{1mc} \subseteq \mathcal{C}$.

The structure of $\mathcal{R}^{1mc}$ depends on the kinematic model of a specific mobile robot. As mentioned in Subsection 4.2.3, the paths obtained under the execution of a single motion command are approximated by arcs of circle or the straight segment. On a circular path, the robot orientation is constrained

by:

$$\theta = \begin{cases} atan(\frac{2xy}{x^2+y^2}), & |x| \geq |y| \\ \pi - atan(\frac{2xy}{x^2+y^2}), & |x| < |y| \end{cases} \tag{4.3}$$

which is expressed in the robot frame of reference.

Equation 4.3 is a **holonomic equality constraint**. The effect is to reduce the dimension of the Configuration space by one [35]. Therefore, the *Reachable Set of a single motion command* $\mathcal{R}^{1mc}$ (i.e. the set of all the configurations that can be reached by the straight segment or one arc of a circle) is a function of only two parameters, and can be represented by $\mathbb{R}^2$. A configuration in $\mathcal{R}^{1mc}$ is represented by $\mathbf{q^{1mc}} = (x, y)$ (once (x, y) are selected, θ is given by Equation 4.3.). Fig. 4.2 depicts how any point (x, y) of the space can be reached, but only with one arc of a circle or the straight segment. The robot orientation at each point (x, y) must be tangent to the arc of circle.

A point worth mentioning here is the fact that both, the robot Workspace, $\mathcal{W}$, and the *Reachable Set of a single Motion Command*, $\mathcal{R}^{1mc}$, can be described by $\mathbb{R}^2$.

For reactive navigation, the sensory information (obstacle information), that is usually given in the Workspace $\mathcal{W}$, still needs to be mapped into the *Reachable Set of a single Motion Command* $\mathcal{R}^{1mc}$.

The obstacles are represented in $\mathcal{W}$ by a set of points $(x, y) \in \mathbb{R}^2$. This is a reasonable assumption since the sources of information are real sensors (e.g. the lasers measure points), and the environment is assumed to be unstructured (e.g. avoiding the use of polygons). Fig. 4.7d and Fig. 4.8d illustrate the type of sensory information that is being dealt with. Then, each obstacle point in $\mathcal{W}$ is mapped into a region in $\mathcal{R}^{1mc}$, called $\mathcal{C}$-*Obstacle* [35]. The union of the $\mathcal{C}$-*Obstacles* is called the $\mathcal{C}$-*Obstacle region*, $\mathcal{R}^{1mc}_{obs} \subset \mathcal{R}^{1mc}$. Fig. 4.8a depicts a differential-driven rectangular robot and the sensory information in $\mathcal{W}$. The $\mathcal{R}^{1mc}$ is illustrated in Fig. 4.8b. Notice that while $\mathcal{C}$ for this robot is three-dimensional and so $\mathcal{C}_{obs}$, the $\mathcal{R}^{1mc}$ and $\mathcal{R}^{1mc}_{obs}$ are two-dimensional.

Appendix B presents an algorithm to compute a discretization of the $\mathcal{R}^{1mc}_{obs}$ boundary. To calculate the exact boundary remains an open problem. The interest is in a discretized version because: (1) in the following, a transformation $\mathbb{R}^2 \rightarrow \mathbb{R}^2$ is presented. Then other representations of the boundary might be discretized to apply the transformation, or the transformation might be redesigned. And (2) the sensory information remains in the form of points, that is the input of most of the existing reactive navigation methods.

Summarizing, the robot Workspace, $\mathcal{W}$, and the subset of the Configuration space needed, the *Reachable Set of a single Motion Command*, $\mathcal{R}^{1mc}$,

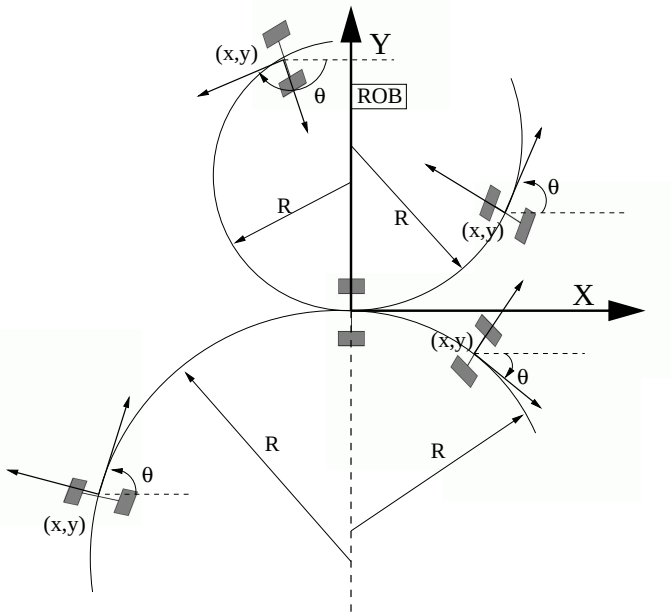


Figure 4.2: The robot orientation, θ, is constrained at any point of the space. The robot moves on arcs of circle, or on the straight segment.

both can be described by $\mathbb{R}^2$. Moreover, a point obstacle in the Workspace is $(x, y) \in \mathbb{R}^2$. The point maps into the *Reachable Set of a single Motion Command*, $\mathcal{R}^{1mc}$, as a region whose boundary can be approximated by points $(x_i, y_i) \in \mathbb{R}^2$. The *obstacle information* in both spaces ($\mathcal{W}$ and $\mathcal{R}^{1mc}$) is fully represented by a set of points in $\mathbb{R}^2$. As a consequence, a coordinate transformation over $\mathbb{R}^2$ can be applied either to $\mathcal{W}$ or to $\mathcal{R}^{1mc}$.

In the following the *Ego-Kinematic Transformation* is presented. This transformation maps $\mathbb{R}^2$ into the *Ego-Kinematic Space*, where the kinematic constraints are implicitly represented.

4.4 The Ego-Kinematic Space

The *Ego-Kinematic Space* (*EK-space*) is constructed by mapping points of $\mathbb{R}^2$ into: (1) descriptors of admissible paths leading to these points, and (2) the

distances to reach these points measured over the admissible paths. This is done by means of the *Ego-Kinematic Transformation*. Since the kinematic constraints are embedded in the *Ego-Kinematic Transformation*, the admissible paths are mapped into straight lines in the transformed space. Then each point of the *EK-space* is reached by a straight line motion "free-flying behavior".

To introduce the *Ego-Kinematic Transformation* (EKT), the attention focuses on the robots discussed in Section 4.2: under the execution of a single motion command they either move over one arc of circle, or over the straight segment.

The family of admissible paths (circles) is first characterized. Considering a point $(x, y) \in \mathbb{R}^2$ in the robot frame, there is only one circle with radius R that verifies: (1) the (x, y) point is on the circle, (2) the origin $(0, 0)$ is on the circle, and (3) the circle center is on the y-axis (the robot instantaneous turning center has to be on the y-axis). The radius of this circle is given by $R = \frac{x^2+y^2}{2y}$ ($R < 0$ represents circles with the center in $y < 0$, and $R = \infty$ is the straight segment). Fig. 4.2 depicts this fact.

The EKT maps each point expressed in the robot frame of reference into the *EK-space*, which is represented in polar coordinates for convenience. This mapping is a function of the circular arc length L, and the radius R of the unique circular path leading to that point.

$$\begin{array}{rcl} \text{EKT:}\mathbb{R}^2 & \rightarrow & \mathbb{R}^+ \times S^1 \\ (x, y) & \rightarrow & (d, \alpha) = (f_d(x, y), f_\alpha(x, y)) \end{array} \tag{4.4}$$

- The distance d is the length of the admissible path leading to the point. Then, d is the arc length L of the circle. From the two arc lengths of the circle (L and $2\pi R - L$), the shortest is selected. This distance is computed as:

$$\begin{array}{rcl} f_d : \mathbb{R}^2 & \rightarrow & \mathbb{R}^+ \\ (x, y) & \rightarrow & d = f_d(x, y) = \left\{ \begin{array}{cc} |R.acos(\frac{x^2-y^2}{x^2+y^2})|, & y \neq 0 \\ |x|, & y = 0 \end{array} \right. \end{array} \tag{4.5}$$

  Fig. 4.3 shows two points (x_1, y_1) and (x_2, y_2) in the robot frame of reference, and the arc lengths $d_1 = L_1$ and $d_2 = L_2$ computed by this function.

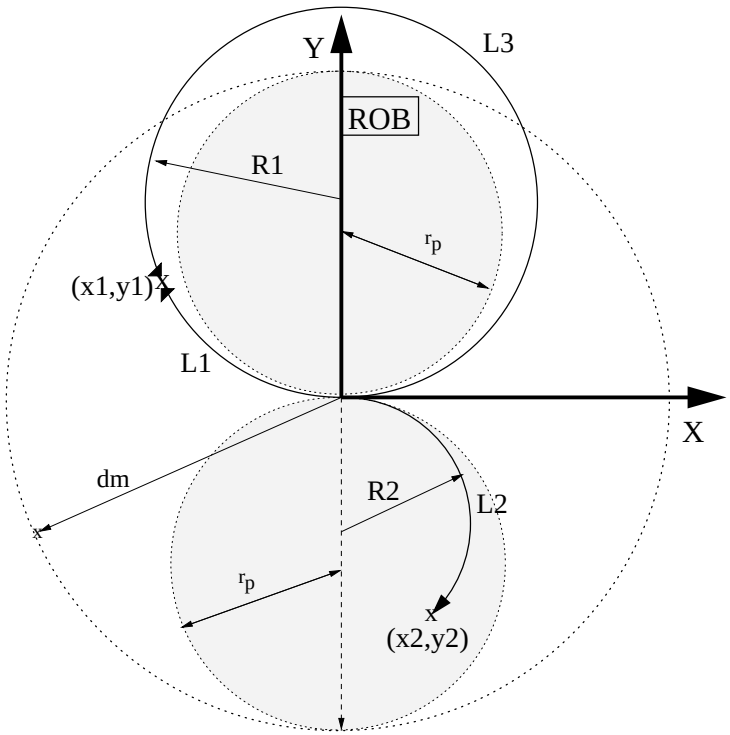


Figure 4.3: Circular paths to reach the points (x_1, y_1) and (x_2, y_2) of the space in the robot frame.

- The angle α parameterizes the family of circular paths arising from the kinematic constraints. The circles family is fully described by the radius $R \in]-\infty, \infty[$. A non-linear function to convert the radius into an angular descriptor is required, in order to have a bounded representation.

  Many functions can be used. Appendix C justifies the selection of the function $\alpha = \arctan(\frac{R}{r_p})$. It converts into an index the relation between a circular path with radius R and the circular path with radius r_p. $r_p = \frac{d_m}{2}$, and d_m is the diameter of the circular region to transform - for practical issues d_m is the distance of the farthest obstacle measured, see Fig. 4.3.

  An arc length, L, does not univocally identify a point on a circle. A different value of α is used to distinguish the traveling direction on a circle:

$$\begin{aligned} f_\alpha : \mathbb{R}^2 &\rightarrow [-\pi, \pi] \\ (x, y) &\rightarrow \alpha = f_\alpha(x, y) = \ \text{sign}(x).\tfrac{\pi}{2} - \arctan(\tfrac{R}{r_p}) \end{aligned} \tag{4.6}$$

Fig. 4.4a shows an example of the EKT applied to a discrete set of points of a polygon. Each side of the polygon is numbered to notice how it transforms into the *EK-space*, illustrated in Fig. 4.4b. The angles $\alpha \in]\alpha_1, \pi - \alpha_1[$ correspond to $0 < R < R_1$, whereas the angles $\alpha \in]\alpha_2, -\pi - \alpha_2[$ result from $0 > R > R_2$. The values of $\alpha = 0$ and $\alpha = \pi$ correspond to both, the straight forward and backward paths respectively along the x-axis.

Some points are worth mentioning here:

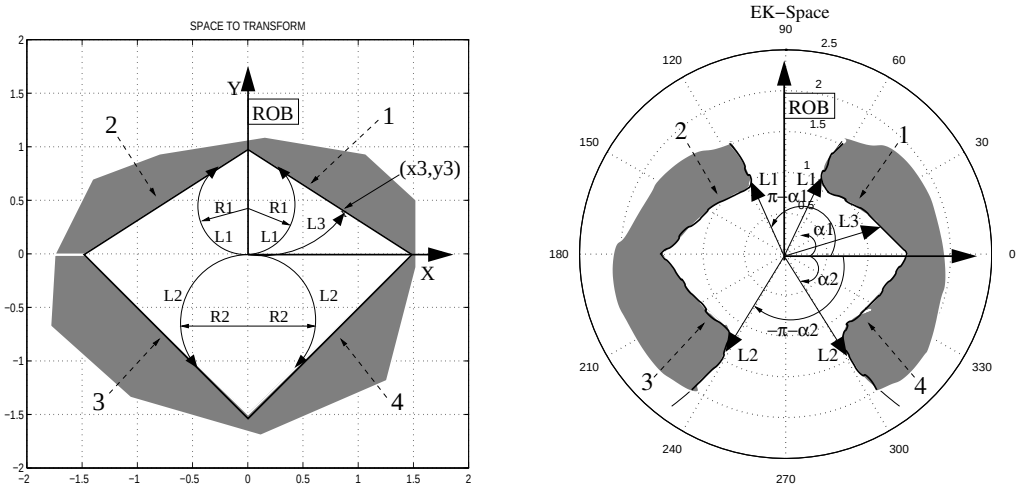
1. The *EK-space* can be computed in closed form.
2. The EKT is invertible. A pair $(d, \alpha) \in$ *EK-space* determines univocally a pair $(x, y) \in \mathbb{R}^2$, and implicitly a $\theta \in S^1$.
3. The distance from the robot to any point in the *EK-space* is adequately described by the Euclidean distance. While this distance is the length of the admissible path that complies with the kinematic constraints.
4. The robot in the *EK-space* is a "free flying object", because it can move in any direction. Notice that a direction of motion α in the *EK-space* settles a unique turning radius for the robot, having $R = f_\alpha^{-1}(\alpha)$.

Next, the *Ego-Kinematic Transformation* is extended to consider the case of robots that can only move forwards. Subsequently, the case of vehicles with curvature constraints is analyzed.

4.4.1 Forward Motion Constraint

The EKT transforms each point into descriptors that univocally describe the path leading to that point. So far circular paths have been considered, where forward and backward motion were allowed. Here, the additional constraint to forbid backward motion is added. This is incorporated to the EKT by imposing a traveling direction over a circle. Fig. 4.3 depicts the idea: to reach (x_1, y_1), L_3 is computed, instead of L_1 that would be calculated by the standard EKT.

The *Ego-Kinematic Transformation* then becomes:



(a) (b)

Figure 4.4: a) Space to be transformed. b) *EK-space*.

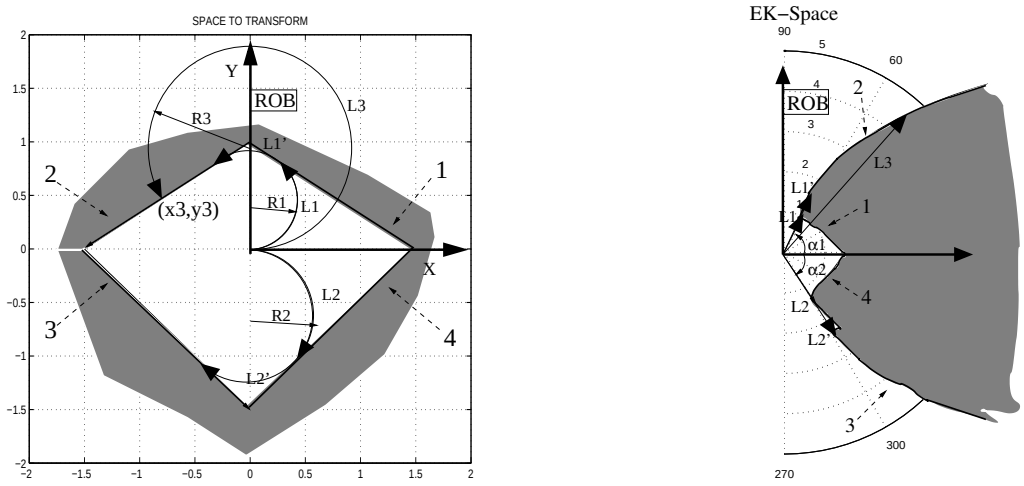
$$\begin{aligned} \text{EKT:}\mathbb{R}^2 &\rightarrow \mathbb{R}^+ \times [-\frac{\pi}{2}, \frac{\pi}{2}] \\ (x,y) &\rightarrow (d = f_d^F(x,y), \alpha = f_\alpha^F(x,y)) \end{aligned} \tag{4.7}$$

where:

$$\begin{aligned} f_d^F : \mathbb{R}^2 &\rightarrow \mathbb{R}^+ \\ (x,y) &\rightarrow d \end{aligned}$$

$$d = f_d^F(x,y) = \begin{cases} |R.acos(\frac{x^2-y^2}{x^2+y^2})|, & x > 0,\ y \neq 0 \\ 2.\pi.|R| - |R.acos(\frac{x^2-y^2}{x^2+y^2})|, & x < 0,\ y \neq 0 \\ x, & x \geq 0, y = 0 \\ \infty, & x < 0, y = 0 \end{cases} \tag{4.8}$$

$$\begin{aligned} f_\alpha^F : \mathbb{R}^2 &\rightarrow [-\frac{\pi}{2}, \frac{\pi}{2}] \\ (x,y) &\rightarrow \alpha = f_\alpha^F(x,y) = \ \text{sign}(y).\frac{\pi}{2} - \arctan(\frac{R}{r_p}) \end{aligned} \tag{4.9}$$



(a) (b)

Figure 4.5: a) Space to be transformed. b) *EK-space*.

Fig. 4.5a shows an example of the EKT applied to a discrete set of points of a polygon. Fig. 4.5b depicts the *EK-space*. The angles $\alpha > \alpha_1$ correspond to $0 < R < R_1$, while the angles $\alpha < \alpha_2$ result from $0 > R > R_2$. The angle $\alpha = 0$ corresponds to the straight forward path. The value of d diverges when $x < 0$ and $y \to 0$, because the radius of the circle leading to that points tends to infinity (the robot cannot move backwards).

4.4.2 Maximum Curvature Constraint

The car-like robot is a platform that has a minimum turning radius R_{min} (or a maximum curvature constraint γ_{max}). This can be incorporated in the EKT by a variable change $R' = R - R_{min}$. The space with $|R| < R_{min}$ is not transformed (as any point within the region $|R|$ cannot be reached under the execution of a single motion command, there is no need to transform it).

Summarizing, this Section presents the *Ego-Kinematic Space*. The *Ego-Kinematic Transformation* is defined over $\mathbb{R}^2$. This result can be used to transform the obstacle information in both, the Workspace $\mathcal{W}$ and in the *Reachable Set of a single Motion Command* $\mathcal{R}^{1mc}$, into the *EK-space*. In this space the kinematic constraints are implicitly embedded, so the robot moves free of kinematic constraints. Thus, the *EK-space* represents the obstacle information in a space where the robot is a "free-flying object".

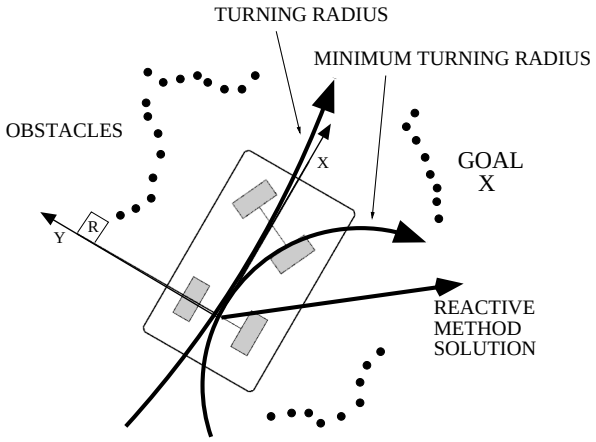


Figure 4.6: The reactive navigation method solution cannot be executed by the non holonomic robot.

Moreover, the *Ego-Kinematic Transformation* has been extended to considered the kinematic constraints of some widely used mobile platforms: either robots that can only move forward, or systems with a minimum turning radius.

The sensor-based motion planning with kinematic constraints is addressed in the following Section. The EKT is used to map the sensory information into the *EK-space*, that implicitly represents the kinematic constraints - the robot is not constrained anymore. Then, the sensor-based motion planning problem is addressed in the *EK-space* without further concern about the kinematics.

4.5 Using the *EK-space* for Sensor-Based Motion Planning

The goal of this Section is to understand how the *EK-space* is used as a framework to address the kinematics in sensor-based navigation. Furthermore, this Section validates experimentally the proposed research.

Many reactive navigation methods assume that the robot is a "free flying object" - they do not address the kinematic constraints. Without taking the robot kinematic constraints into account, the execution of the motion computed by a reactive method is doomed to rely on some approximations. The Fig. 4.6 depicts this fact. The robot cannot execute the method solution, and the motion that complies with the kinematic constraints is forward. Instead, the *EK-space* is used to implicitly represent the robot kinematic constraints.

Reactive navigation methods not designed for non-holonomic robots can be applied to the *EK-space*. (Irrespective whether the method applies to the Workspace or to the Configuration space.) Finally, the reactive navigation method solution in the *EK-space* is transformed back to the Workspace, to compute an admissible motion command.

At each sample period T, the procedure to use the *EK-space* is the following:

1. The sensory information is reduced to points expressed in the robot frame of reference.

   (a) If the reactive method applies to the Workspace $\mathcal{W}$, the EKT is directly applied to the obstacle points.

   (b) Or, if the reactive method applies to the Configuration space, the $\mathcal{C}$-obstacle region $\mathcal{R}_{obs}^{1mc}$ is first built (as described in Section 4.2), to apply the EKT.

   In both cases the result is the obstacle information expressed in the *EK-space*.

2. The reactive navigation method is then applied to the *EK-space* to compute the most promising motion direction, α.

3. A turning radius is calculated by $R = EKT^{-1}(\alpha)$. Finally, a motion command that preserves this turning radius is obtained by $v = \omega.R$.

Any strategy to compute a pair (v, ω) that preserves R is valid. In the current implementation v is calculated with the distance to the closest obstacle. Then, ω is computed. If $\omega > \omega_{max}$, then $\omega = \omega_{max}$ and v is computed preserving the turning radius R.

To validate this methodology experimentation with two different reactive navigation methods has been conducted. Both methods do not take the kinematic constraints into account. The reason to select the methods is to cover almost all the results presented in the Chapter: firstly the Nearness Diagram Navigation, that applies to the Workspace and only allows forward motion. Secondly, the Potential Field Method, that applies to the Configuration space and allows both forward and backward motion.

Notice that the results obtained are not compared with other methods. The reason is that the scope of the Chapter is the spatial representation, not the reactive method itself. The particular results are completely dependent on the reactive method used, and its specific implementation.

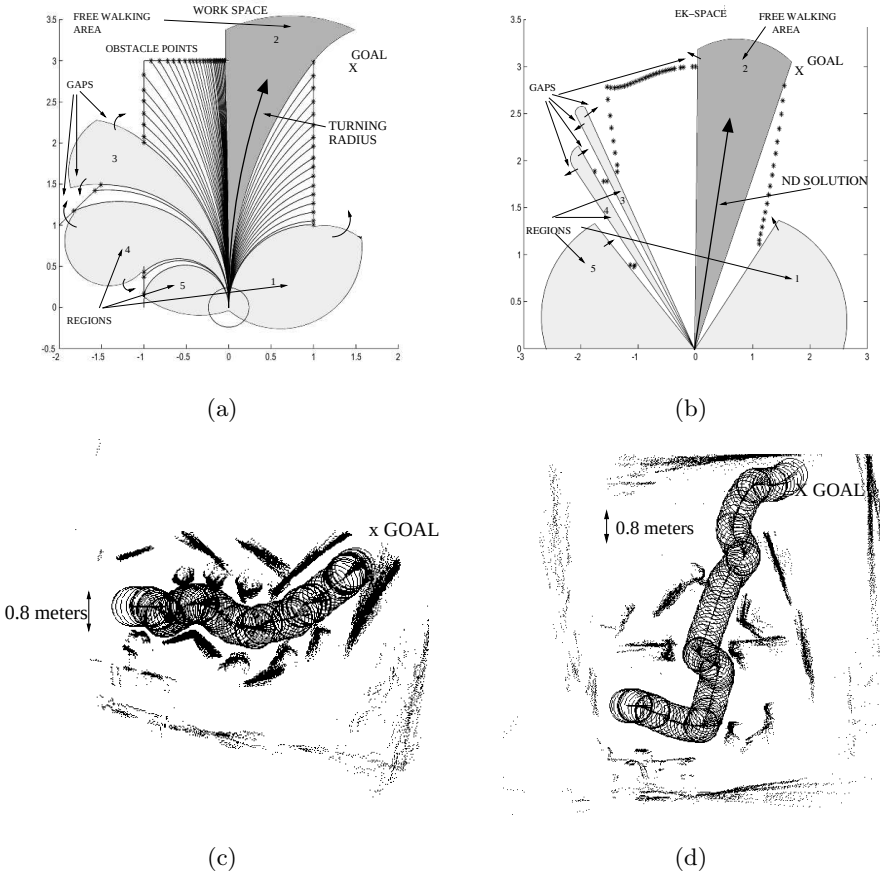


Figure 4.7: a)Workspace . b)*EK-space*. c) Experiment. d) Experiment.

4.5.1 Platform and Experimental Settings

The framework was tested in a *Labmate* robot at the University of Zaragoza, Spain. The main sensor is a 3D *TRC* laser rangefinder. For further details about the robot and sensor, see Appendix E.0.5.

In all the experiments the environment was completely unknown, and only the goal location was given in advance to the robot. The environments were unstructured (random shaped obstacles). Moreover, the environment could be dynamic and non-predictable. Under all these circumstances it is widely justified the use of sensor-based motion planning algorithms to move the robot.

4.5.2 Nearness Diagram Navigation

The *Nearness Diagram Navigation (ND)*, Chapter 2, is a reactive navigation method that does not take into account the kinematic constraints.

The ND method is described in detail in Chapter 2. However, a brief description of the method is next presented. The ND method is based on the situated-activity methodology of design [3]. First, a set of five situations that fully describe the relative state of the robot, obstacle distribution, and goal location are defined. Subsequently, one action is designed for each situation. In real-time, the sensory information is used to identify the current situation, and the associated action is executed computing the motion commands.

To identify the current situation, in the sensory information some high level information is searched for: gaps, regions, free walking areas. Some criteria are also applied: a security criterion, a free walking area criterion, etc. The individual action designs are based on a geometrical implementation.

To apply the ND method to robots with kinematic constraints the *EK-space* is used.

At each sample period T the following procedure is repeated:

1. The EKT is applied to the sensory information in the Workspace (Fig. 4.7a). The result is the obstacle information in the *EK-space* (Fig. 4.7b).

2. The ND method is applied to the *EK-space* to compute the most suitable motion direction α^* (represented in Fig. 4.7b as the ND solution). Some information used by the ND to compute the solution (the gaps, regions, and the free walking area, see Fig. 4.7a) is shown in Fig. 4.7b.

3. The direction α^* is transformed back to the Workspace as a turning radius $R^* = EKT^{-1}(\alpha^*)$, see Fig. 4.7a. Finally, a motion command $(\mathbf{v}, \mathbf{w})$ that preserves this turning radius is computed.

This procedure was tested on the real robot. The cycle time of the algorithm (construction of the *EK-space* and reactive method usage) is around $150msec$ on a $100MHz$ microSun SparcII. This execution time is enough for real-time. Fig 4.7d shows a real experiment. The robot was safely driven to the goal location - the only information given a priori to the robot. A circular robot is drawn since a circle approximates the robot shape (due to ND implementation requirements). Notice that the robot is safely driven among closed obstacles. The reason is that the ND method alone has been demonstrated to successfully drive platforms in dense, cluttered, and complex scenarios, see Chapters 2. The usage of the *EK-space* to address the kinematic constraints

does not penalize the method advantages. Thus, the complete framework is still able to deal with such troublesome scenarios.

4.5.3 Potential Field Approach

Potential Field Methods (PFM) [30] used as a reactive navigation method do not take into account the kinematic constraints.

The robot is considered as a particle in the Configuration space, moving under the influence of an artificial potential field. While the goal creates a potential that attracts the particle, the obstacle information creates a repulsive potential. The motion is computed to follow the direction of the artificial force induced by the sum of the two potentials.

This framework cannot be used with non-holonomic robots. The reason is that the gradient direction of the potential does not preserve the non-holonomic constraint of Equation (4.1). In other words, in the Configuration space not all differential motions are allowed. This is not represented by the potential structure.

The PFM can be applied to non-holonomic robots using the *EK-space*. The following procedure is repeated at each sample period:

1. The obstacle points (sensory information) in the Workspace, Fig. 4.8a, are used to construct the $\mathcal{C}$-Obstacle region $\mathcal{R}_{obs}^{1mc}$ in the *Reachable Set of a single Motion Command*, see Fig. 4.8b. Then, the EKT is applied to $\mathcal{R}_{obs}^{1mc}$, yielding the obstacle information in the *EK-space*, see Fig. 4.8c.

2. The PFM is applied to the *EK-space* to compute the most promising motion direction, α^*, represented in Fig. 4.8c as the PFM solution.

3. The direction, α^*, is transformed back to the Workspace as a turning radius (see Fig. 4.8a), determined by $R^* = EKT^{-1}(\alpha^*)$. Finally, a motion command $(\mathbf{v}, \mathbf{w})$ preserving this turning radius is computed.

This procedure was tested on the Labmate platform. The following expressions were used for the goal and repulsive forces [35], $\mathbf{F_g}$ and $\mathbf{F_r}$:

$$\mathbf{F_g}(\mathbf{q}) = -K_g(\mathbf{q} - \mathbf{q}_g) \tag{4.10}$$

$$\mathbf{F_r}(\mathbf{q}) = \begin{cases} K_r(\frac{1}{d(\mathbf{q})} - \frac{1}{d_0})\frac{1}{d(\mathbf{q})^2}\frac{\partial d(\mathbf{q})}{\partial \mathbf{q}} & if\ d < d_0 \\ 0 & otherwise \end{cases} \tag{4.11}$$

where $\mathbf{q} = (x, y)$, K_g and K_r are constants, $d_g(\mathbf{q})$ and $d(\mathbf{q})$ are the distances from the robot to the goal and obstacle locations. The *clearing distance* is

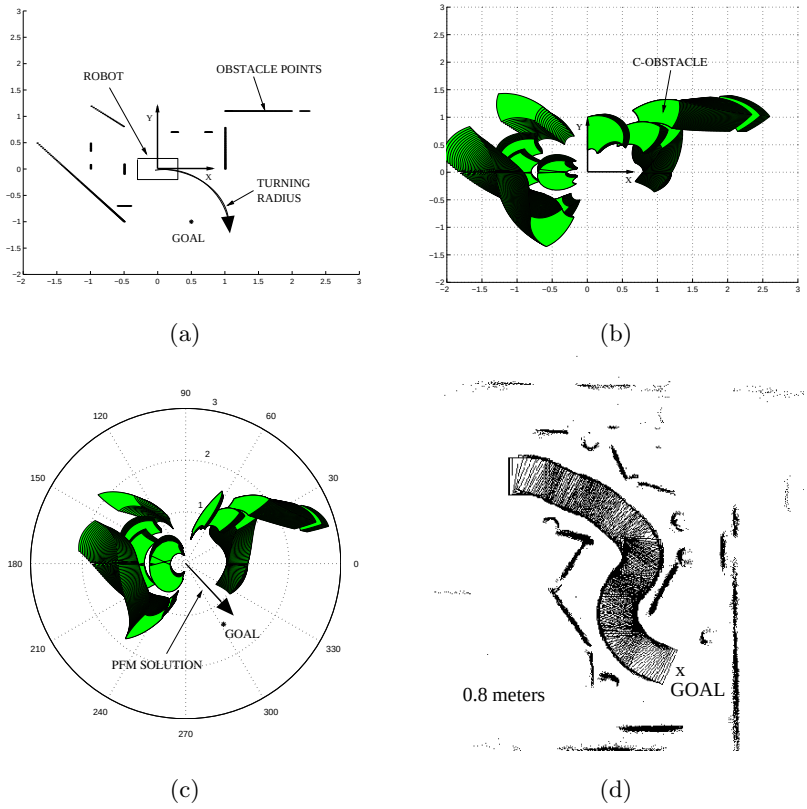


Figure 4.8: (a) Workspace. (b) *Reachable Set of a single motion command* $\mathcal{R}_{obs}^{1mc}$. (c) *EK-space*. (d) Experimental result.

denoted by d_0. $\frac{\partial d(\mathbf{q})}{\partial \mathbf{q}}$ represents the unit vector pointing away from the $\mathcal{C}$-Obstacle region, and that is supported by the line passing through $\mathbf{q}$ and the obstacle point of the region.

Notice that the original potential formulation deals with the Configuration space, that is $\mathbb{R}^2 \times S^1$. Here, the potential is defined in the *Reachable Set of a single Motion Command*, that is $\mathbb{R}^2$. The subspace of the configuration needed has one dimension less (as detailed in Section 4.3).

The execution time of the algorithm (construction of the *Reachable Set of a single Motion Command*, *EK-space*, and reactive method usage) is around $250msec$. Hence this cycle-time was enough for real-time. Fig. 4.8d shows an experiment performed with the real platform. The complete framework was able to drive the constrained platform free of collisions to the goal location -

the only information given a priori to the robot. The experiment highlights the effect of the *EK-space* to address the kinematics: arcs of circle mainly compose the motions computed and executed.

4.6 Conclusions

This Chapter addresses the sensor-based motion planning with kinematic constraints. The novelty of the idea is to use the motion constraints to construct a spatial representation - *Ego-Kinematic space*. The kinematic constraints are implicitly represented in this space, so the robot moves as a "free-flying object". Then, it is straightforward to use in this space existing reactive navigation methods. Therefore, the method solutions comply with the kinematic constraints. A wide solution to address the kinematic constraints in the sensor-based motion planning discipline has been presented.

The Configuration space for the robots covered in the analysis is $\mathbb{R}^2 \times S^1$. Another contribution is to show that only a subspace of the Configuration space, $\mathbb{R}^2$, is needed (in the context of sensor-based motion planning). An algorithm to calculate the bounds of the $\mathcal{C}$-Obstacle region in this subspace is given. The author thinks that this result will alleviate the computations required for reactive navigation methods in the future.

The proposed research has been validated experimentally with a real platform. By using the methodology, two reactive navigation methods were used to successfully drive the constrained platform among locations. Both methods are not originally formulated to work on non-holonomic robots.

Extensions to other non-holonomic robots are straightforward. The admissible paths that arise from their kinematic constraints would derive in new formulations of the *Ego-Kinematic Transformation*.

This research addresses the kinematic constraints, but ignores the effect of the dynamics. We have been working in a similar approach described in detail in Chapter 5: to introduce the dynamic constraints in the space. Our objective is to merge both works to address the sensor-based motion planning with kinematic and dynamic constraints.

Chapter 5

Sensor-Based Motion Planning with Dynamic Constraints

5.1 Introduction

The Robotic community is building robots for many fields and applications. Many of these applications require the safe motion generation: Service robots, demining robots, healthcare robots, and supervision robots, among others. Some of these robots exhibit dynamic constraints due to the mechanical construction, or to the environment of application: robots with limited torques, robots working in non-gravity conditions, underwater robots, and blimps, among others. Moreover, some applications require moving the platforms at relatively high-speeds. In both cases the dynamic capabilities of the robot take an important place, constraining the feasible system motions.

This Chapter presents a framework to take into account the robot dynamic constraints in the reactive collision avoidance layer. The proposal is to use the dynamic constraints to build a spatial representation, where the dynamic constraints are implicitly represented. With minor modifications, many existing reactive navigation methods, that do not take the dynamic constraints into account, can be applied to this space. As a consequence, the motion commands computed comply with the robot dynamics. This is an ample solution to incorporate the dynamics into much of the current work developed in sensor-based motion planning.

The novelty is that the dynamic constraints are implicitly embedded in the

spatial representation where the reactive heuristic applies. This differs this research from, related work that address dynamics by modeling or identifying the vehicle behavior, or other researches that redesign a reactive method from scratch to address this issue.

Some advantages derive from the proposed framework, irrespective of the reactive navigation heuristic. The motion commands computed comply with the robot dynamics - the vehicle can execute the planned motion. The vehicle trajectories that arise from the motion commands are collision-free. Moreover, the motion commands are *Secure* for the platform - the guaranty for safely stopping the robot if it is required always exists. By addressing the dynamics, the performance and safety of the reactive method that makes use of the framework is improved - without redesigning the reactive method itself.

To demonstrate the utility of this framework, two reactive collision avoidance approaches have been extended to address the dynamic constraints: the Potential Field method [30] and the Nearness Diagram Navigation (Chapter 2). Both methods have been tested on a circular and holonomic platform. The experiments are reported to validate the framework.

This research was previously presented in [46].

The Chapter is organized as follows. Section 5.2 and 5.3 present related work and the dynamics covered in the analysis. Sections 5.4 and 5.5 introduce the *Ego-Dynamic Space* and *Spatial Window*. Section 5.6 combines both concepts to build the complete framework. Section 5.7 presents the combination of the framework with reactive navigation methods. The experimental results are discussed in Section 5.8, and in Section 5.9 the conclusions are drawn.

5.2 Related Work

This Section presents related work to sensor-based motion planning. The attention focuses on the reactive navigation methods.

The reactive navigation methods are commonly accepted to generate safe motion in unknown, unstructured, and dynamic environments. This field of research has been extensively studied in the last decade. Some researchers solve the reactive navigation problem making a physical analogy: Potential Field methods [30], [34], [58], [5]; the perfume method [6], the fluid methods [43]. Other authors attempt the reactive navigation problem by computing a set of motion commands, to select the "better" later on: the Vector Field Histogram [11], and the successors [59], [60]. Other researches solve the reactive navigation problem by calculating some high-level devices, that allow for computing a motion command: Elastic Band [54], Elastic Strips [14], and the

Nearness Diagram Navigation (Chapter 2).

Most of the reactive navigation methods do not address dynamics. Thus, these methods are susceptible to failure in the case analyzed in this Chapter. The dynamic constraints have been mainly addressed in sensor-based motion planning from two different points of view: (1) some researches deal with dynamics by modeling the system behavior. Some of them model the system [31], [5], [16]. Others identify the system model by the responses to motion commands (inputs) [2], [40]. Once the model is available, the system responses are also known, and used to apply reactive navigation strategies. (2) Some authors explain the system response with a model of constrained inputs. Some of them solve the reactive navigation problem in the motion command space as a constrained optimization [56], [25], [13]. Others compute dynamic admissible trajectories, to obtain the motion commands later on [22], [62].

The spaces where the reactive navigation methods usually apply are the Workspace $\mathcal{W}$, [44], [14], [50], [6], [43], [11]; or the Configuration space $\mathcal{C}$ [35], [30], [34], [58], [59], [60], [54]. The research presented here is based on a spatial representation - prior to the reactive method usage - where the system dynamics are implicitly represented. Then, both spaces, $\mathcal{W}$ and $\mathcal{C}$, are analyzed to achieve the maximum generality.

5.3 Dynamics in Motion Commands

This Section first analyses the vehicle case of study. Subsection 5.3.2 presents the motion commands considered, and in Subsection 5.3.3 the dynamics involved in the motion command execution are discussed.

5.3.1 Vehicle

The vehicle considered is a circular and holonomic robot moving on a flat surface with the classical hypothesis "rolling without slipping". The Workspace $\mathcal{W}$ is $\mathbb{R}^2$. The Configuration space $\mathcal{C}$ is $\mathbb{R}^2 \times S^1$. Since the robot is circular, the Configuration space can projected in $\mathbb{R}^2$. Then both, the Workspace and the Configuration space are represented by $\mathbb{R}^2$. The space of motion commands is three-dimensional, where $(\mathbf{v}, \mathbf{w})$ is a motion command. $\mathbf{v} = (v_x, v_y)$ is the translational velocity, and $\mathbf{w}$ is the rotational velocity. Both velocities expressed in the robot reference system. The attention focuses on $\mathbf{v} = (v_x, v_y)$ because it modifies the robot location (the robot rotational velocity, $\mathbf{w}$, is not dependent on the translational velocity).

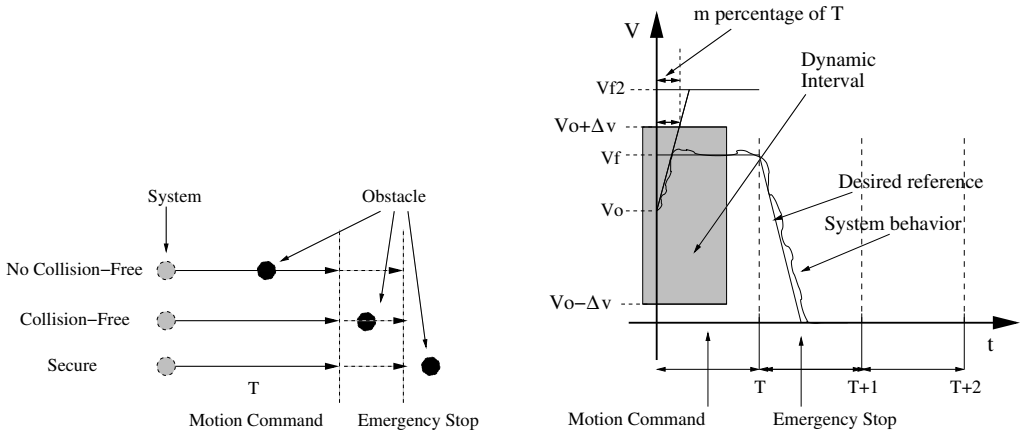


Figure 5.1: a) Types of motion commands. b) Motion command execution.

5.3.2 Motion Commands in Reactive Navigation

Based on a perception-action process, the reactive navigation methods compute at each time the *"best" motion command*: to avoid collisions whilst moving the robot towards a given goal location. The types of motion commands considered are next summarized:

- *Emergency Stop*: this command is a policy to stop the robot by applying the maximum system deceleration.
- *Collision-Free commands*: the execution of these motion commands is free of collisions during the next sample period T.
- *Secure commands*: these motion commands ensure: (1) the execution is collision-free during the next sample period T - they are *Collision-Free*, and (2) after the execution of the motion command, the robot can be stopped with the *Emergency Stop* without colliding.

Fig. 5.1a depicts the motion commands in the one-dimensional case. The system executes a motion command, and later, the *Emergency Stop* is launched. The first motion command is not *Collision-Free*. The second motion command is *Collision-Free*. But notice that after the command execution, the robot cannot avoid the collision. The last motion command is a *Secure command*. This command produces a *Collision-Free* motion during T, and later, the robot can

be safely stopped an *Emergency Stop*. In reactive navigation the objective is to compute this type of motion commands: *Secure commands*.

Since the objective of this chapter is to introduce the dynamic constraints in the reactive navigation methods, the next Section presents the dynamics involved in the motion command execution process.

5.3.3 The Dynamic Constraints

The robot acceleration constrains the motion, when the robot controller executes motion commands. The controller design is not addressed here. However, the controller behavior must be taken into account to deal with the dynamic constraints.

There are the two dynamic constraints determined by the maximum acceleration-deceleration, a_b, of the system:

1. *Braking constraint*: It appears when the *Emergency Stop* is launched. The controller is designed to apply the maximum deceleration, a_b, to stop the robot, see Fig 5.1b.

2. *Dynamic interval*: The controller is designed to reach the steady state of a reference command, v_f, as soon as possible. First, the maximum acceleration of the system, a_b, is applied to reach the reference velocity. Subsequently, the steady state is maintained, see Fig. 5.1b. Given the current robot velocity, v_o, the commands within the dynamic interval $v_{next} \in [v_o \pm \triangle v]$ are *dynamic admissible*. To compute $\triangle v$, the assumption is that the system can instantaneously reach the desired velocity - the robot moves at a constant velocity during the sampling period T. Then, $\triangle v = |a_b|.m.T$ (m is the sampling period T percentage that the system spends to reach the steady state). The position error between the real system behavior, and the constant velocity assumption is $e_d = \frac{|a_b|.(m.T)^2}{2}$. If e_d is out of our requirements, a lower m is selected. Thus, $\triangle v$ is reduced.

The first constraint sets the maximum distance that the robot travels when the *Emergency Stop* is launched. The second constraint determines the set of dynamic admissible motion commands that can be selected for motion.

The design of the spatial transformation, that embeds the deceleration constraints into the spatial representation, is presented next.

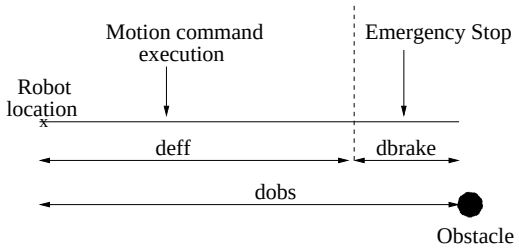


Figure 5.2: a) Distance to an obstacle.

5.4 The Ego-Dynamic Space

The rational idea is to build a spatial representation. The distances to the obstacles are transformed into distances that depend on the deceleration constraint of the robot, and on the sampling time. In this space - *Ego-Dynamic Space* (*ED-Space*) - the first dynamic constraint presented in Subsection 5.3.3 is represented.

The analysis is firstly carried out in one dimension, see Fig. 5.2. d_{obs} is the measured distance from the robot to an obstacle. d_{eff} is the *effective distance*: the maximum distance that the robot travels at a constant velocity during the time period T, and that later allows the *Emergency stop* for stopping the robot without collide (applying the maximum deceleration a_b).

$$\begin{aligned} d_{obs} &= d_{eff} + d_{brake} \\ d_{eff} = v.T \qquad d_{brake} &= \frac{v^2}{2.a_b} \\ \frac{d_{eff}^2}{2.a_b.T^2} + d_{eff} - d_{obs} &= 0 \end{aligned} \tag{5.1}$$

where d_{eff} is obtained, and thus the *Ego-Dynamic Transformation* (EDT):

$$\begin{array}{rcc} \text{EDT:}\mathbb{R}^+ & \rightarrow & \mathbb{R}^+ \\ d_{obs} & \rightarrow & d_{eff} = a_b.T^2.(\sqrt{1+\frac{2.d_{obs}}{a_b.T^2}} - 1) \end{array} \tag{5.2}$$

This distance, d_{eff}, is a motion constraint: if the robot moves farther than d_{eff}, the robot will not be safely stopped with the *Emergency Stop*. d_{eff} depends on: (1) the measured distance to the obstacle, d_{obs}. (2) The

deceleration capabilities of the robot, a_b. (3) The sampling period, T, in which the motion command is applied.

In reactive navigation sometimes the deceleration effects are neglected - infinite deceleration capabilities are assumed. This is represented by Equation (5.1): when $a_b \rightarrow \infty$, d_{eff} tends to be the measured distance d_{obs} used by these methods.

This framework extends to two dimensions with an upper bound error: the braking trajectory (parabola) is not a straight line, as assumed in the one-dimensional case. This error can be neglected in the context of reactive navigation (deeper details are discussed in Appendix D). Then:

$$\begin{array}{ccc} \text{EDT:}\mathbb{R}^2 & \rightarrow & \mathbb{R}^2 \\ (x,y) & \rightarrow & (d_{eff_x}, d_{eff_y}) \end{array} \tag{5.3}$$

This spatial representation is called the *Ego-Dynamic space*. The *Ego-Dynamic Transformation* can be applied to both, the Workspace and the Configuration space, leading the obstacle information to the *Ego-Dynamic Space*. Next, the advantage of the ED-space is highlighted by an example:

Example 3 *Fig. 5.3 shows an example of the* EDT *applied to both, the $\mathcal{W}$ and $\mathcal{C}$, to study the effect of the maximum deceleration, a_b, in the spatial representation. Fig. 5.3a illustrates the robot in the Workspace and the current perception (laser scan). $a_b = 1.0\frac{m}{sec^2}$ and $T = 0.5sec$. The EDT is applied to the obstacle points in $\mathcal{W}$, leading the obstacle information to the ED-space (see Fig. 5.3b), which is expressed in the robot frame of reference. To transform $\mathcal{C}$, first the C-Obstacles are computed by enlarging each obstacle point with the robot radius. Then, the EDT is applied. The result is the obstacle information in the ED-space, see Fig. 5.3c. Comparing the Workspace, Fig. 5.3a, and the ED-space, Fig. 5.3b, the obstacle information is closer to the robot when the dynamic constraints are taken into account. The ED-space fully represents the dynamic capability: the obstacles are taken into account before, than the case where the dynamics are neglected, that is, the obstacles are represented closer to the robot. Next, the dynamic capabilities of the robot are reduced, $a_b = 0.5\frac{m}{sec^2}$. The resulting* ED-space *from the Configuration space is illustrated in Fig. 5.3d. Comparing Figs. 5.3c,d the obstacles are closer to the robot when the robot has less dynamic capabilities.*

The advantage of this spatial representation is that the robot deceleration capabilities and sample period are implicitly represented in the space. The next Section addresses the second mentioned constraint.

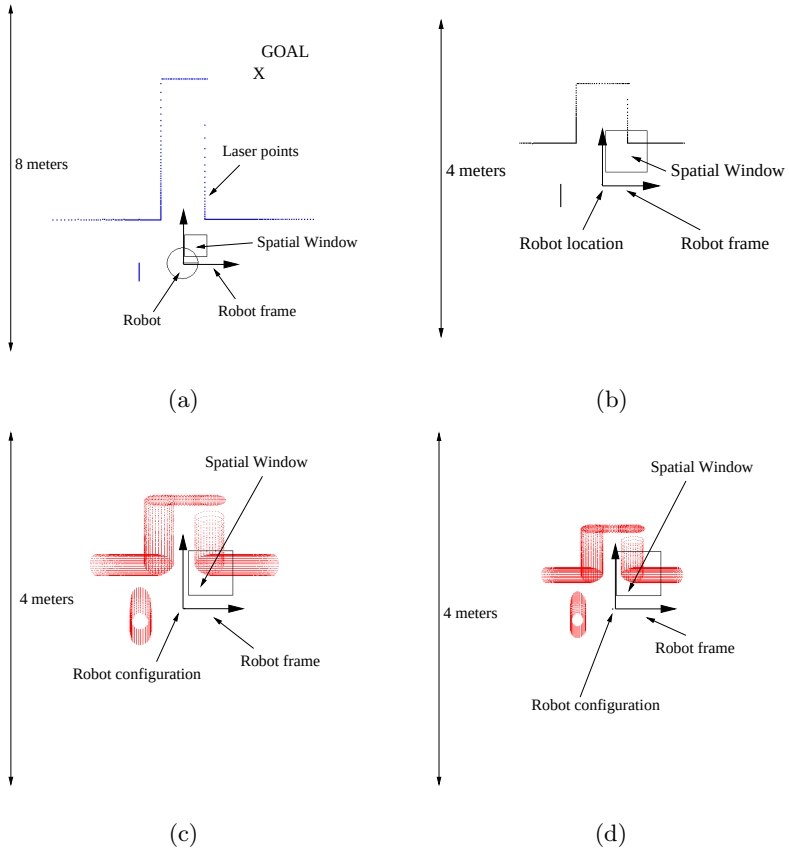


Figure 5.3: a) Workspace. b) ED-space from the Workspace ($a_b = 1.0\frac{m}{sec^2}, T = 0.5sec$). c) ED-space from the Configuration space ($a_b = 1.0\frac{m}{sec^2}, T = 0.5sec$). d) ED-space from the Configuration space ($a_b = 0.5\frac{m}{sec^2}, T = 0.5sec$)

5.5 The Spatial Window

This Section presents the *Spatial Window* to deal with the second of the dynamic constraints presented in Subsection 5.3.3: the next motion command has to be within the dynamic interval $[\mathbf{v_o} \pm \triangle\mathbf{v}]$.

The *Spatial Window* (SW) is the set of robot configurations obtained under the execution of motion commands within the dynamic interval. Assuming a constant velocity during the period T, the corners of the *Spatial Window* are given by:

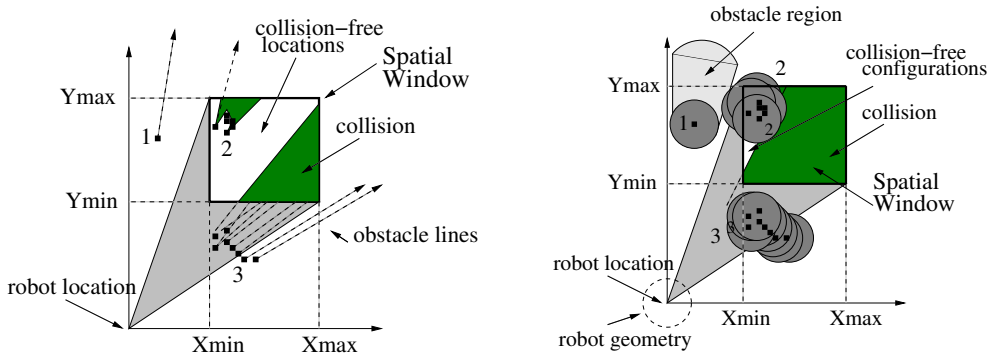


Figure 5.4: a) Spatial Window in the Workspace. b) Spatial Window in the Configuration space.

$$X_{max} = (v_{ox} + \triangle v).T \quad and \quad X_{min} = (v_{ox} - \triangle v).T$$
$$Y_{max} = (v_{oy} + \triangle v).T \quad and \quad Y_{min} = (v_{oy} - \triangle v).T$$

where $\mathbf{v_o} = [v_{ox}, v_{oy}]$ is the current translational velocity, see Fig. 5.4a.

The *Spatial Window* is used to represent the possible collision-free locations in the Workspace $\mathcal{W}$ and in the Configuration space $\mathcal{C}$. Figs. 5.4a,b show examples of the SW in both spaces. In the Workspace $\mathcal{W}$, Fig. 5.4a, each obstacle point creates an *obstacle line* of locations that cannot be reached under the execution of a single motion command without collisions. The intersection of these *obstacle lines* with the SW gives the collision locations inside the SW. There are three cases: (1) the point is outside the SW, and the *obstacle line* does not intersect the SW, see point 1 in Fig. 5.4a. (2) The point is inside the SW, see point 2 in Fig. 5.4a. The segment of the *obstacle line* inside the SW fixes the collision locations. (3) The point is outside the SW, but the *obstacle line* intersects the SW, point 3 in Fig. 5.4a. The same strategy used previously. In the Configuration space $\mathcal{C}$, Fig. 5.4b, the obstacle points create the $\mathcal{C}$-Obstacles. The procedure is the same, but each $\mathcal{C}$-Obstacle produces an *obstacle region* instead of an *obstacle line*, see Fig. 5.4b. In both cases, the SW contains the locations or configurations that can or cannot be reached without collisions (under the execution of a single motion command within the admissible dynamic interval).

Once a collision-free location, $\mathbf{x_p} = (x_p, y_p)$, inside the *Spatial Window* is calculated, a *Collision-Free command* is computed by $\mathbf{v} = (\frac{x_p}{T}, \frac{y_p}{T})$. The

motion command, $\mathbf{v}$, is *dynamic admissible* by the system, because $\mathbf{v}$ is within the admissible dynamic interval. This ensures feasible motion **execution**.

The objective of this work is to take into account the dynamics in the motion generation layer. Moreover, the motion commands computed have to be *Secure commands*, as mentioned in the previous Sections. The next Section combines the *Spatial Window* and the *Ego-Dynamic Space*, to comply with both requirements.

5.6 The Framework

This Section unifies in a framework the *Ego-Dynamic Space* and *Spatial Window*: (1) calculating *Secure commands*, to assure robust motion commands **calculation**. (2) taking into account the dynamics involved in the motion command generation, to ensure feasible motion commands **execution**.

At each sampling period T, the procedure is the following:

1. The obstacle information is reduced to points expressed in the robot frame of reference.

   - To transform the Workspace, $\mathcal{W}$, the EDT is applied to the obstacle points.
   - If not, to transform the Configuration space, $\mathcal{C}$, first $\mathcal{C}$-Obstacle region is calculated. Then the EDT is applied.

   In both cases the result is the obstacle information expressed in the ED-space.

2. The SW is directly applied to ED-space. The SW locations or configurations that cannot be reached due to the obstacle distribution are labeled in collision.

3. Any strategy to select one collision-free location, $\mathbf{x_p}$, inside the SW is valid. Then, a motion command is computed by $\mathbf{v} = (\frac{x_p}{T}, \frac{y_p}{T})$. The next Section presents how to use reactive navigation methods to achieve this goal.

Two examples that highlight the relevance of this framework are discussed next:

Example 4 *Figs. 5.3a,b,c show this framework on a robot with a fast dynamic capability working at high speeds (maximum acceleration* $a_b = 1.0\frac{m}{sec^2}$*). The sample*

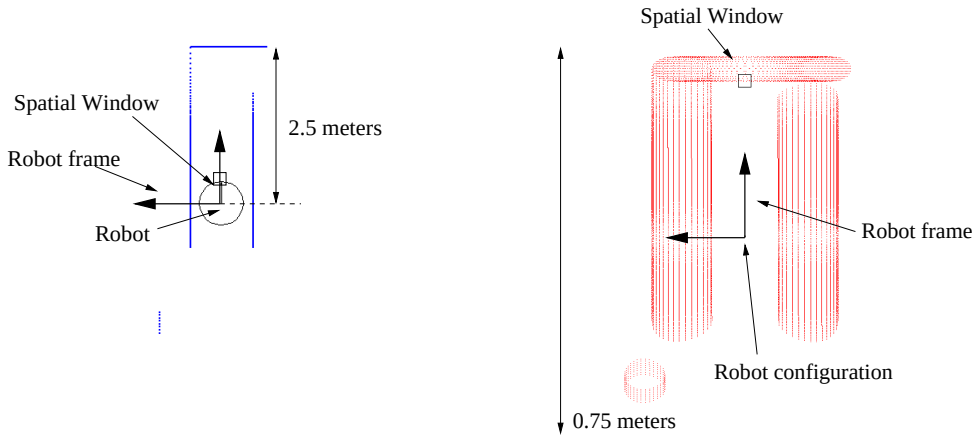


Figure 5.5: a) Robot moving in the Workspace. b) ED-space from $\mathcal{C}$.

period is $T = 0.5sec$. *Fig. 5.3a depicts the robot moving with* $\mathbf{v_o} = (0.6\frac{m}{sec}, 0.8\frac{m}{sec})$ *that fixes the SW in the space. The SW is free of obstacles, thus all locations produce* Collision-Free *motion commands for the next sample period. On the other hand, by applying the EDT to* $\mathcal{W}$ *or* $\mathcal{C}$*, the obstacle information leads to the ED-space, see Figs. 5.3b,c. In both cases the ED-space represents the obstacles closer to the robot due to the dynamic capabilities. Thus, there are obstacles inside the SW that constrain the locations that can be chosen. Once a collision-free location inside the SW in the ED-space is selected, a* Secure command *is computed. Notice that without considering the dynamics in this situation, the robot might crash into the corner.*

Example 5 *An example with a system with slow dynamics,* $a_b = 0.1\frac{m}{sec^2}$*, is illustrated in Fig. 5.5. The robot is in the same environment presented in Fig. 5.3a, but located in the center of the corridor, facing the frontal wall, and moving at* $\mathbf{v_o} = (0.61\frac{m}{sc}, 0.0\frac{m}{sc})$ *towards this wall* $(2.5m)$*. In the ED-space, Fig 5.5b, the obstacles are very close to the robot location, due to the slow dynamics. Moreover, there are some obstacles inside the SW. The collision-free locations inside the SW are those closer to the robot. These locations will produce motion commands that reduce the velocity of the robot. This is the correct behavior: the robot starts to stop before expected to avoid the collision, even if the obstacle is located far away,* $2.5m$*. Neglecting the dynamics in this case, the robot will crash the frontal wall.*

In this framework all locations inside the SW can be in collision. Then, the *Emergency Stop* is launched to safely stop the robot. Subsequently, the motion is resumed.

The main interests of the - *ED-space + SW* - framework are:

1. The motion commands are *Secure commands*, because they are computed from locations in the ED-space.

2. The motion commands comply with the system dynamics. They are within the dynamic interval $\mathbf{v_{next}} = [\mathbf{v_o} \pm \triangle \mathbf{v}]$, because the SW is used.

The next Section presents one strategy to select a location inside the SW, implicitly fixing the motion command.

5.7 Combining the Framework and the Reactive Navigation Methods

The framework presented in the previous Section leaves open the problem of selecting one location inside the SW. This Section presents a solution based on reactive navigation methods.

Notice that any strategy to select one collision-free location inside the SW solves the problem. [25], [13], and [56] solve a similar problem by a constrained optimization that balances the goal location, the forward progress, and the obstacle clearance. Similar strategies could be used.

The solution here is to use existing reactive navigation methods in the ED-space, to select a collision-free location inside the SW. The reasons for choosing these techniques are:

1. The presented framework has to work in real-time. The usage of a reactive method does not impose a significant time penalty. Everything -*ED-space + SW + Reactive method* - can be run in real-time.

2. The reactive navigation methods take directly into account the goal location; and information about the obstacle spatial distribution - this advantage is lost when the problem is solved with other types of heuristics, e.g. with a constrained optimization as [25], [13], [56].

3. The reactive navigation method is a module not dependent on the rest of the framework. Different reactive methods can be used and tested within the framework without significant changes.

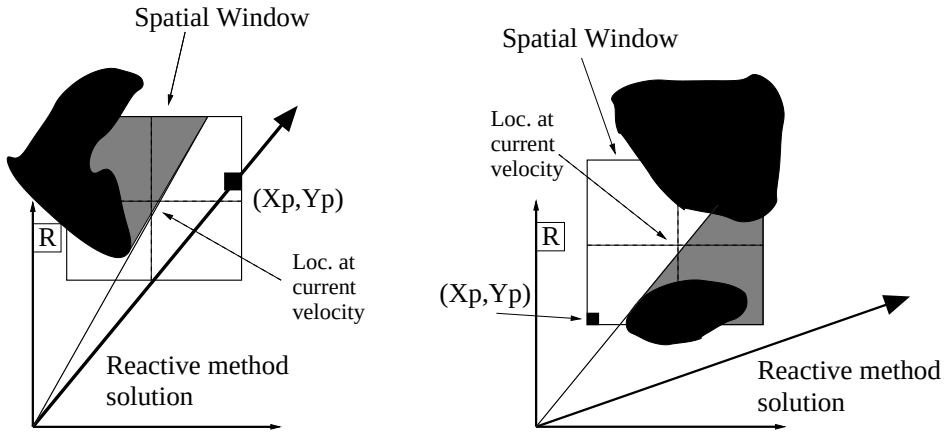


Figure 5.6: a) The location solution inside the SW.

The solution of most reactive navigation methods is a *most promising motion direction* [30], [50], [11], [59], [54], [14], [44], among others. The strategy is to apply the reactive navigation method to the ED-space. Then, the direction solution is used to select a collision-free location inside the SW. The location is selected as follows:

1. The direction solution intersects the SW (the robot can move towards the direction that reactive method is computing). The location (inside the SW) over the direction solution, and farthest from the robot location is selected. This heuristic favors the maximum forward progress, while moving over the reactive method solution. See Fig. 5.6a.

2. The direction solution does not intersect the SW (the robot cannot move towards the direction that reactive method is computing). The closest location (inside the SW) to both, the robot location and the direction solution is selected. This heuristic reduces the robot velocity, while bringing the robot closer to the reactive method solution. See Fig. 5.6b.

Finally, the motion command is computed by $\mathbf{v} = (\frac{x_p}{T}, \frac{y_p}{T})$.

5.8 Experimental Results

This Section validates experimentally the proposed research. Experiments were conducted with two reactive navigation methods: the Nearness Diagram Navigation [44], that applies to the Workspace $\mathcal{W}$. Secondly, the Potential Field Method [30], that applies to the Configuration space $\mathcal{C}$. Both methods do not take into account the dynamic constraints. By using the proposed framework, the vehicle dynamics are directly addressed.

Notice that the experimental results obtained are not compared to other methods. The scope of the Chapter is the complete framework - *ED-space + SW + Reactive method*, and not the reactive method itself. The particular results are completely dependent on the reactive method used and its specific implementation.

However, the difference of the each reactive method behavior, when using or not the proposed framework, is discussed. The benefits of using the reactive method with the framework are: (1) the motion commands computed are *Secure motions*: they are collision-free in execution, and they give the guaranty for safely stopping the robot without collisions. (2) The motion command are dynamic feasible. Both benefits are not dependent on the navigation heuristic used. Then, the comparison is not directed in quantitative terms, since the usage of the framework improves the method behavior. The comparison is directed in qualitative terms: it was tested how far the fact of ignoring dynamics, produces motions that are not the planned ones - collision avoidance cannot be guaranteed. This comparison demonstrates the robustness and quality of the approach. Moreover, other issues as global execution time are discussed.

The same environment, robot, and sensor were used for the method comparison. However, it is very difficult to have the same navigation results in two different trials, since real systems are used.

For experimentation was used a *Nomadic XR4000* at *LAAS-CNRS*, France. The robot is equipped with a *SICK* 2-D laser rangefinder. See Appendix E.0.4 for a deeper description of the robot and sensor.

5.8.1 Nearness Diagram Navigation

The *Nearness Diagram Navigation (ND)* [44] is a reactive navigation method that does not take into account the dynamic constraints.

The ND method is described in detail in Chapter 2. However, a brief introduction to the method is next presented. The ND method is based on the situated-activity methodology of design [3]. First, a set of five situations that fully describe the relative state of the robot, obstacle distribution, and goal

location are defined. Subsequently, one action is designed for each situation. In real-time, the sensory information is used to identify the current situation, and the associated action is executed computing the motion commands.

To identify the current situation, some high level information is searched for in the sensory information: gaps, regions, free walking areas. And some criteria are also applied: a security criterion, a free walking area criterion, etc. The individual action designs are based on a geometrical implementation.

Using the proposed framework, the ND is used taking into account the dynamic constraints. At each sample period the following procedure is repeated:

1. The EDT is applied to the sensory information in the Workspace (laser scan). The result is the obstacle information in the ED-space. (Procedure presented in Section 5.4)

2. The SW is filled using the obstacle information in the ED-space. (Procedure presented in Section 5.5)

3. The ND method is applied to the ED-space to compute the most suitable motion direction α^*. This direction α^* is used to select a collision-free location, $\mathbf{x_p}$, inside the SW. (Procedure presented in Section 5.6). Finally, a motion command is computed by $\mathbf{v} = (\frac{x_p}{T}, \frac{y_p}{T})$.

The procedure was tested on the real platform. Fig. 5.7a shows the result of one of the runs. The robot successfully reached the goal location while avoiding collisions with the environment. Figs. 5.7c,d depict the behavior of the system during the experiment: the motion commands reference computed by the framework (v_x and v_y), and the real ones executed by the robot (for better appreciation only a fraction of the experiment is shown). Notice that the system can execute the computed commands (the velocity error is illustrated in Fig. 5.7e). The consequence is that the trajectories carried out are near to the planned ones: by addressing dynamics, robust collision avoidance is guarantied. On the other hand, the same experiment was carried out with the reactive method alone (ND). The velocity profile is illustrated in Fig. 5.7f,g. The system could not correctly execute the computed commands (the velocity error is illustrated in Fig. 5.7h. The consequence is that the trajectories carried out are distant from the planned ones: collision avoidance can be no longer guaranteed. More experiments were run to characterize the velocity error profile, see Fig. 5.7b. The error distribution gives the idea of the error in the executed trajectories.

Notice that the robot is safely driven among close obstacles, Fig. 5.7a. The reason is that the ND method alone has been demonstrated to successfully

drive platforms in dense, cluttered, and complex scenarios, see Chapter 3. The usage of the ED-space to address the dynamic constraints does not penalize the method advantages. Thus, the complete framework is still able to deal with troublesome scenarios.

From a global point of view, using the framework improves the safety of the method: the trajectories executed are near to the computed ones. Beyond, the framework gives the guaranty for safely stopping the robot if required. The drawback is the global execution time. The same experiments took about two or three times longer that the ones without the dynamic considerations. The reason is that the framework is very conservative. Only *Secure* velocities are computed. They usually penalize the acceleration because safety is prioritized.

5.8.2 Potential Field Method

Potential Field Methods (PFM) [30] can be used as reactive navigation methods, but they do not take the dynamic constraints into account.

The robot is considered as a particle in the Configuration space, moving under the influence of an artificial potential field. While the goal creates a potential that attracts the particle, the obstacle information creates a repulsive potential. The motion is computed to follow the direction of the artificial force induced by the sum of the two potentials.

Using the proposed framework, the PFM are used taking into account the dynamic constraints. The following procedure is repeated at each sampling period:

1. The sensory information in the Workspace is used to construct the $\mathcal{C}$-Obstacle region. Then, the EDT is applied to the $\mathcal{C}$-Obstacles, leading the obstacle information to the ED-space.

2. The SW is filled using the obstacle information of the ED-space.

3. The PFM is applied to the ED-space to compute the most promising direction of motion. This direction is used to select a collision-free location, $\mathbf{x_p}$, inside the SW. Finally, a motion command is computed by $\mathbf{v} = (\frac{x_p}{T}, \frac{y_p}{T})$.

Fig. 5.8a shows the result of one run with the real platform. Figs. 5.8c,d depict the motion commands reference computed by the framework (v_x and v_y), and the real ones executed by the robot (for better appreciation only a fraction of the experiment is shown). The velocity error is illustrated in Fig. 5.8e. The same experiment was repeated with the reactive method alone

(PFM). The velocity profiles are illustrated in Fig. 5.8f,g, and the velocity error in Fig. 5.8h. Some experiments were repeated to characterize the velocity error profile (see Fig. 5.8b). The difference in between both experimentations (with and without the framework) is less outstanding than with the ND. The PFM implementation is based on a function that changes smoothly in the space. This results in smooth changes in the motion commands computed (accelerations), and so in the errors neglecting the dynamics. However, after repeating the experiment, the error distribution is still better when dynamics are not neglected (Fig. 5.8b).

Although in the experiments without dynamic considerations the navigation is accomplished, it cannot be always ensured. The conclusion is that using the proposed framework the robot executes in any case the velocity set points planned. This results in safe and feasible motions, but higher execution time.

5.9 Conclusions

This Chapter addresses the problem of applying reactive navigation methods to systems where the dynamic constraints cannot be neglected: (1) systems working at high-speeds, or (2) systems with slow dynamics.

The advantage of the framework is that the system dynamics are directly taken into account. Moreover, the computed motion commands are *Secure commands*: (1) they are *Collision-Free* during the execution time, and (2) they give the guaranty for safely stopping the robot with an *Emergency Stop* policy.

The - *ED-space* and *Spatial Window* - have been presented as a general framework to take into account the dynamics of the systems in sensor-based motion planning. To validate the research, experimentation with two reactive methods that do not consider dynamics has been presented. By using the framework, both methods successfully achieve navigation taking into account the robot dynamics.

The presented framework extends to the third dimension with generality. This will allow the reactive navigation methods that operate in three dimensions to make the most of the presented framework. The assumption is a circular and holonomic vehicle. The author believes that the future work will be to merge the presented research with the research presented in Chapter 4. This will allow to use the framework for systems with non-circular shapes, and with kinematic and dynamic constraints.

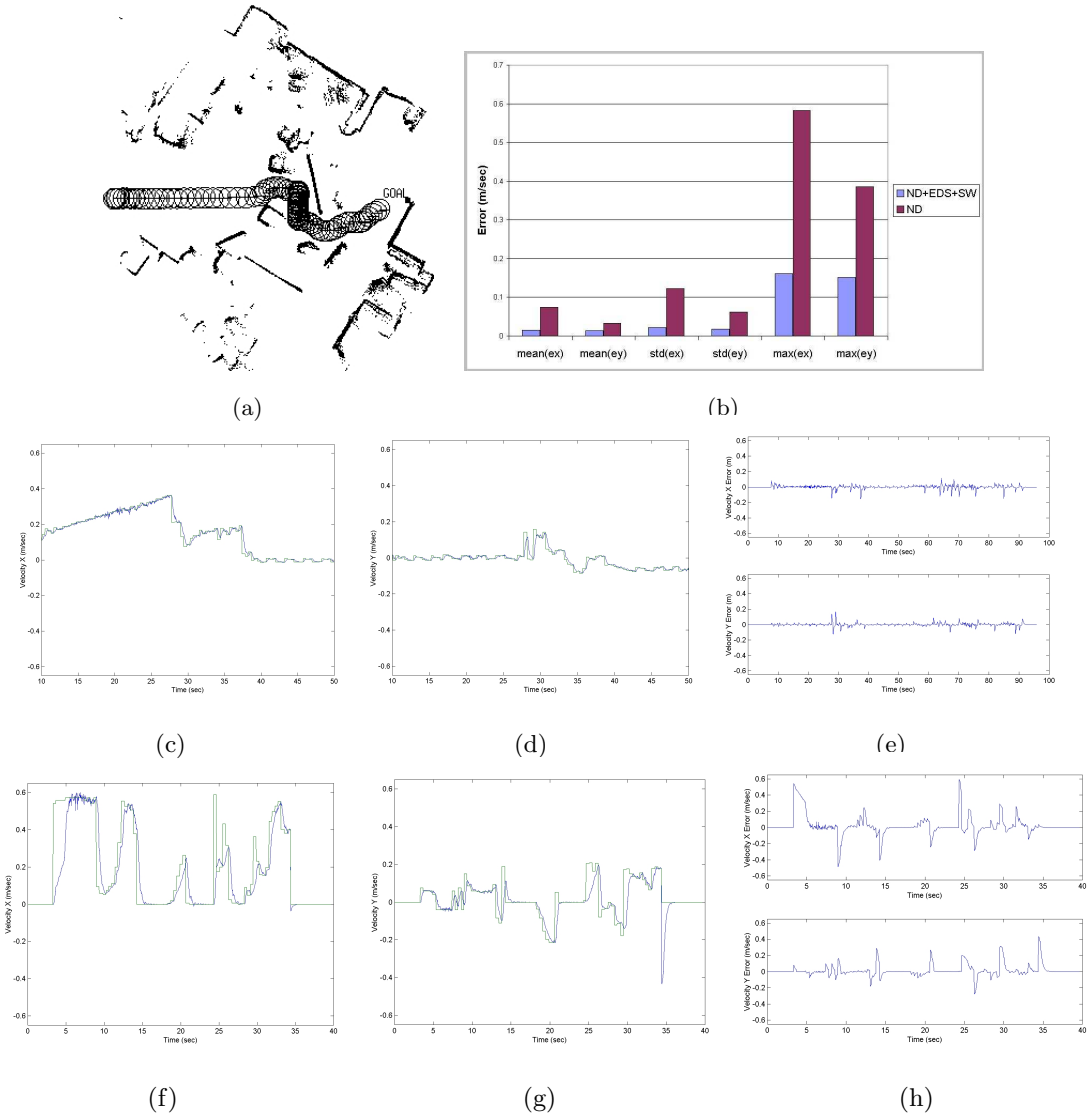
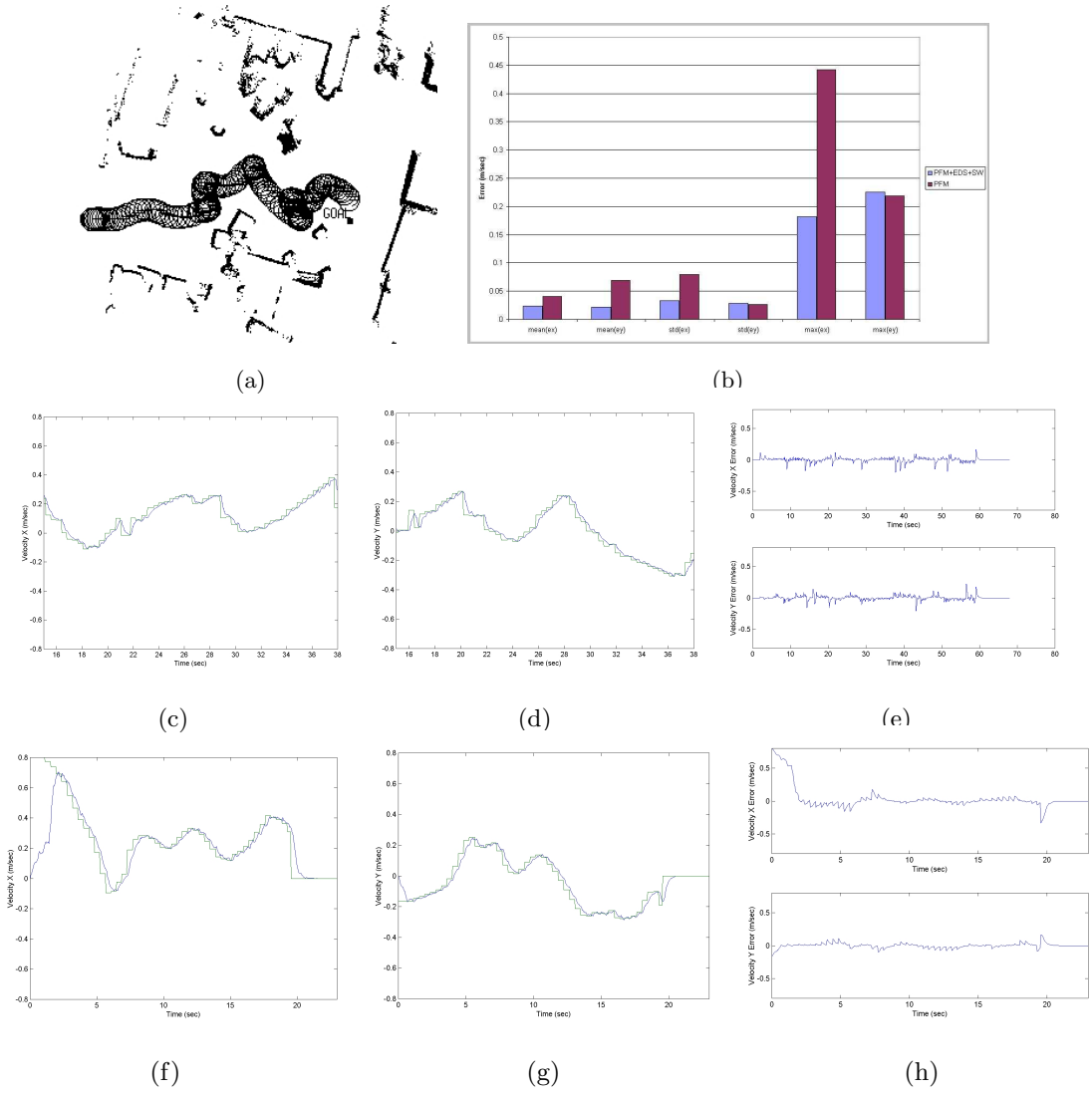


Figure 5.7: a) Experiment the ND. b) Velocity error distribution of the experiment. c,d) Velocity profile of the experiment within the framework. e) Error in the velocity profile. f,g) Velocity profile of the experiment without the framework. h) Error in the velocity profile.



(a) (b)

(c) (d) (e)

(f) (g) (h)

Figure 5.8: a) Experiment the PFM. b) Velocity error distribution of the experiment. c,d) Velocity profile using the framework. e) Error in the velocity profile. f,g) Velocity profile without using the framework. h) Error in the velocity profile.

Chapter 6

The Sensor-Based Navigation System

6.1 Introduction

The Robotic community is building robots to accomplish tasks that require motion in human environments. "Unfortunately", the human environments are designed for humans, and the robot algorithms must adapt to these work places. The robots then deal with any shape unknown obstacles randomly placed in the environment. Moreover, the robot motion algorithms face another difficulty in these environments: they are full of humans that also move. Sharing the environment with humans, converts the work place in a highly dynamic scenario. Besides, the robot must also preserve the human safety (avoiding collisions with humans!). Fig. 6.1 illustrates an scenario of such characteristics. The motion in these environments is a challenge for Robotics, but required for many mobile applications. This problem is dealt with in this Chapter: moving a robot in human-made scenarios that it is highly dynamic.

The Chapter firstly analyses the sensor-based navigation requirements that must be considered in order to navigate in the suggested scenarios. The compliance with the requirements mainly determines the success of the navigation task, thus they all must be considered in the sensor-based navigation design. The proposed sensor-based navigation system is made up of three modules: a *mapping module*, a *planning module*, and a *reactive navigation method*. They all cooperate to complete the collision avoidance task: (1) the *mapping module* deals with the sensory process, and thus with the environment nature - the module is designed to model unknown, highly dynamic, unstructured, and non-predictable scenarios. (2) The *planning module* extracts the workspace

Figure 6.1: a) Robot moving in an office environment.

connectivity. This provides the system with information to avoid problems related to the environment structure - avoiding the trap situations and cyclic behaviors in the robot motion. (3) The *reactive navigation method* safely drives the robot in dense, cluttered, and complex scenarios.

The Chapter also proposes an architecture to integrate the modules in real-time. The integration improves the reactivity of the system by exploiting the robustness of the modules, without sacrificing the base functionalities. Besides, module failure detection and recovery are dealt with. The architecture is generic for a sensor-based navigation system. The base functionalities could replaced by other different instances that adapt to the particular application context, robot, or sensor.

Seen as a whole, a robust sensor-based navigation system, that safely moves a robot in human environments, is presented. The proposed framework is not a global navigation system. To solve the complete navigation problem, the proposed system needs to be integrated with other systems such as supervision [51], [15], [32], localization and map building [18], [17], [38], [20], [57], and global path planning [35].

This research was previously presented in [48].

The Chapter is organized as follows: The system requirements are introduced in Section 6.2, and discussed in relation to related work in Section 6.3. Section 6.4 presents the three module-functionalities design. Section 6.5 introduces the complete navigation system. Section 6.7 discusses the framework and in Section 6.8 the conclusions are drawn.

6.2 Navigation System Requirements

The aim of this work is to develop a sensor-based navigation system to safely drive a robot among locations in any human scenario. For this reason, the requirements that the environment nature and structure impose upon the navigation system design are next summarized:

1. **Robust reactive navigation**: the motion commands have to be robustly computed for a sensor-based motion planner. The reactive algorithm has to solve highly complex navigation problems: to successfully navigate in very dense, complex, and cluttered scenarios.

2. **Information integration**: past sensory perceptions have to be integrated into a representation of the environment. Two reasons justify this: (1) it gives a framework to have an incremental global reasoning. (2) Information of the past perceptions must be used to avoid obstacles not perceived at the current moment (visibility sensor constraints).

3. **Reaction in dynamic environments**: when the environment changes dynamically in a non-predictable way, the environmental representation has to model rapidly the change. Otherwise, the robot will avoid parts of the space known to be free of obstacles, or will not avoid perceived obstacles.

4. **Trap situations and cyclic behaviors avoidance**: There are many obstacle configurations that produce trap situations, common for all the reactive navigation methods. The most typical are the U-shape obstacles. Moreover, there are some symmetric obstacle distributions where the reactive methods can produce alternated solutions. These environments create cyclic behaviors in the robot motion. The robot will never reach the goal location in both cases, unless a key is supplied.

5. **Functionalities integration**: The functionalities designed need to be integrated within an architecture. Otherwise, interactions among func-

tionalities must be designed from scratch to allow module replacement - portability among different platforms/sensors. Therefore, the module coordination requires failure detection and recovery.

The goal of this Chapter is to present a sensor-based navigation system designed to comply with the above five requirements.

6.3 Related Work

This Section presents related work about sensor-based navigation systems. The discussion is directed towards the five requirements presented: robust reactive navigation, information integration, dynamic environments reaction, trap situations and cyclic behaviors avoidance, and functionalities integration.

Many sensor-based motion planners have been developed in the last years. Some of them use a physical analogy to compute motion commands [30], [34], [58], [9], [50]. Particular solutions to the inherent problems of these methods [33], f.e. [19], still appear in the literature. Other methods first compute some sets of motion commands. Next, a navigation strategy selects one motion command of these sets [22], [28], [56], [25]. Or there are methods that calculate some form of high-level information description from the sensory information. Then, a motion command is computed, as opposed to being selected from a pre-calculated set [54], [14]. The limitation of these methods is to **robustly navigate** in complex, dense, and cluttered scenarios. These scenarios present a high degree of difficulty for many existing sensor-based motion planners.

[14] is a framework that uses a path computed by a path planning algorithm. Incremental adjustments of the path are based on the sensory data, while maintaining the path in the free space. Some strategies were introduced to deal with **dynamic environments**: (1) letting an obstacle to jump over the path, and (2) maintaining a set of alternative routes for selection. **Trap situations** are avoided since a global path is always used. The difficulties of this method are to deal with situations where the path does not exist anymore, and to achieve **robust navigation** in dense, complex, and cluttered scenarios, see [12].

[59] uses an occupancy-grid of the environment, [10], to **integrate the information**. The reactive method uses the obstacle information of the local map to compute safe motion commands. In **highly dynamic environments**, the method presents the difficulty of the sensors used, ultrasounds. Recently, [60] was presented. The **trap situations** are avoided by running the reactive algorithm some steps before the algorithm execution. This method obtains

good results even running on platforms with low computational capabilities: the sensory integration is computationally efficient, and even with a reduced number of steps local trap situations are avoided. However, the convergence to the goal location depends on the number of steps selected. [11], [59], and [60] have the difficulty to achieve **robust navigation** when switching between dense and less dense scenarios, see [59].

[13] is a sensor-based navigation system that formulates the problem as a constrained optimization in the velocity space. The cost function incorporates global information to avoid the **trap situations**. The method uses a global-referred occupancy-grid that represents the robot configuration space. This local map is used to **integrate the sensory information**. The difficulties of this method are the computational load, and updating the free space that is indispensable in **dynamic environments** (the configuration space is represented in the grid).

The **module integration** is a point worth mentioning here. [59], [60] and [13] are sensor-based navigation systems that are made up of modules. However, the module integration is not developed within any kind of architecture.

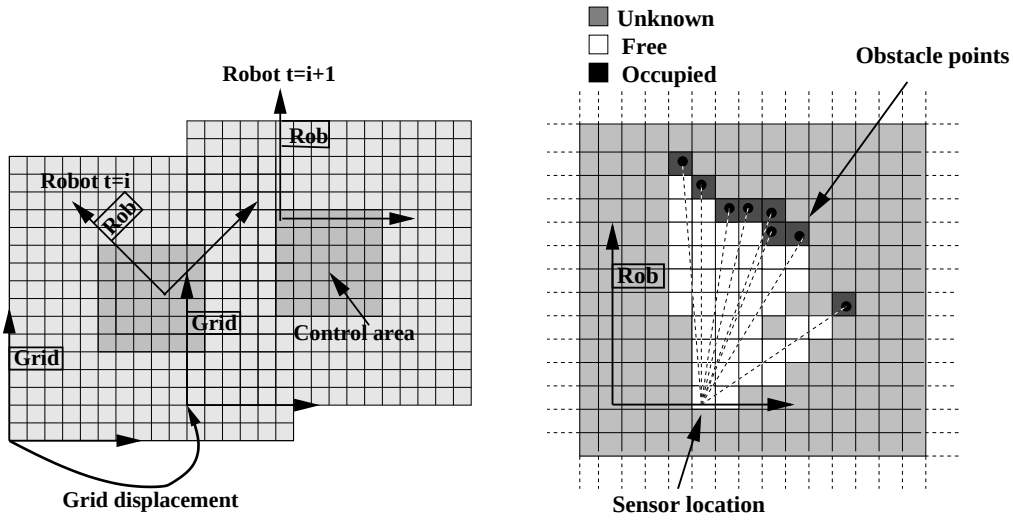
6.4 The Module-Functionalities Design

This Section presents the design of the modules that make up the sensor-based navigation system. Firstly, the reactive algorithm base is discussed, Subsection 6.4.1. Secondly, the design of the module dealing with the sensory process is presented, Subsection 6.4.2. Finally, Subsection 6.4.3 introduces the module that allows for the trap situation avoidance.

6.4.1 Reactive Navigation Module

The reactive navigation method used is the *Nearness Diagram Navigation*, that is described in detail in Chapter 2. However, a brief description is next presented. The ND method is based on the situated-activity methodology of design [3]. First, a set of five situations that fully describe the relative state of the robot, obstacle distribution, and goal location are defined. Subsequently, one action is designed for each situation. In real-time, the sensory information is used to identify the current situation, and the associated action is executed computing the motion commands.

The reason for selecting the ND method is because it achieves **robust navigation**. Good results in *very dense, complex, and cluttered scenarios*



(a) (b)

Figure 6.2: a) Robot motion between two consecutive times, and grid displacement to encompass the robot location. b) Grid update with the laser data.

have been reported using different platforms and sensors, see Chapters 2 and 3. These are the human-made environments where the robot has to move.

So far a reactive algorithm complies with the first requirement of the navigation system has been selected. The sensory process has still to be addressed.

6.4.2 Mapping Module

Five requirements were outlined in Section 6.2 in order to design the sensor-based navigation system. Next, the design of the *mapping module* is presented. The module complies with the second and third requirements: **information integration** and **dynamic environments reaction**.

The navigation is performed in a two dimensional space. The sensor used to measure the environment is a 2D laser range-finder. No assumptions about the environment (static/dynamic, structured/unstructured ...) are made.

The *mapping module* supervises a local model of the environment. The model is constructed by integrating the sensory information in an occupancy-grid, [10], that represents a fraction of the robot workspace.

The robot is the reference of the occupancy-grid. To handle the grid movements, a *control area* is defined around the grid center. When the robot escapes

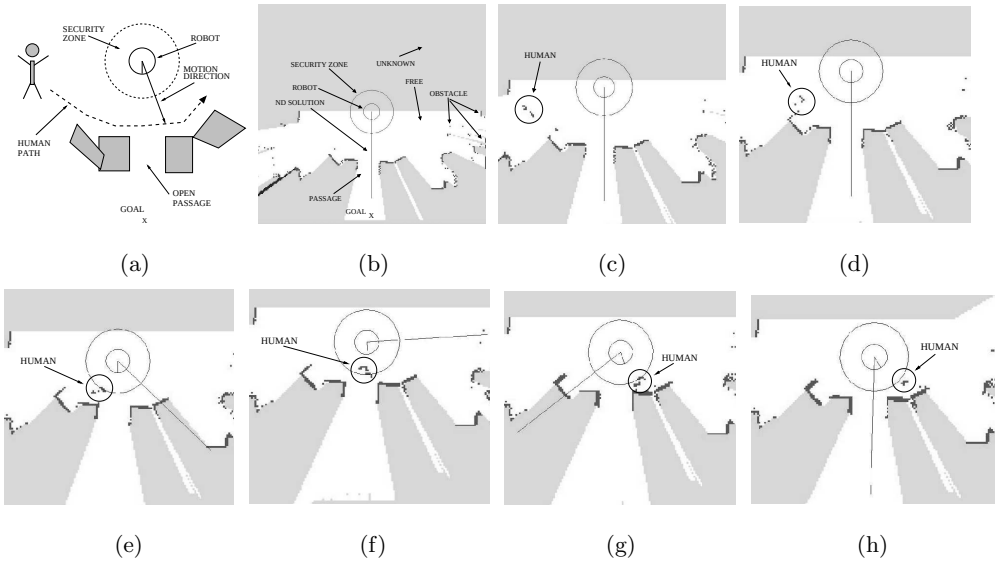


Figure 6.3: Experiment using the *Mapping* and *Reactive Navigation module*.

from the *control area*, the grid is moved to encompass the robot within the *control area*. This allows the robot to move within the *control area*, without having to move the complete grid. The grid displacements are multiple of the cell dimension, and the grid rotation is not allowed. This avoids the dissemination of false obstacle information among cells (remarkable source of errors). Fig. 6.2a illustrates the process of a grid diplacement due to the robot motion. The robot location at time $t = i + 1$ is out of the *control area*. Thus, the complete grid moves to encompass the robot location within the *control area*.

The laser data are placed directly into the grid model without any preprocessing. The occupancy-grid is made up of three types of cells: *occupied*, *free*, and *unknown*. The grid cells coinciding with the obstacle points returned by the laser sensor are labeled *occupied*. The cells between the sensor and each obstacle point are labeled *free* ones. Initially, all the grid cells are labeled *unknown* (never perceived). Fig. 6.2b depicts this process. The Bresenham algorithm [24] is used to optimally implement the above procedure in order to achieve real-time performance.

The framework relies on the fact that the robot surroundings are constantly sensed, and the grid is updated at high rate. Under these conditions, and the reliability of the sensor sensor used, temporal occupancy factors were not added to the grid.

Some points are worth mentioning here:

1. The current perception (laser scan) merged into the occupancy-grid has no drift errors with respect to the robot location. Thus, only the grid cells, not updated with the current perception, accumulate drift errors. Moreover, spurious measures are removed from the grid while new laser scans are integrated. Assuming that little slippage occurs during motion, this model is suitable for **integrating the information** at different times (for obstacle avoidance context).

2. The **dynamic environments** are rapidly reflected in the occupancy-grid. This is a consequence of updating the entire grid area covered by the last perception (laser scan).

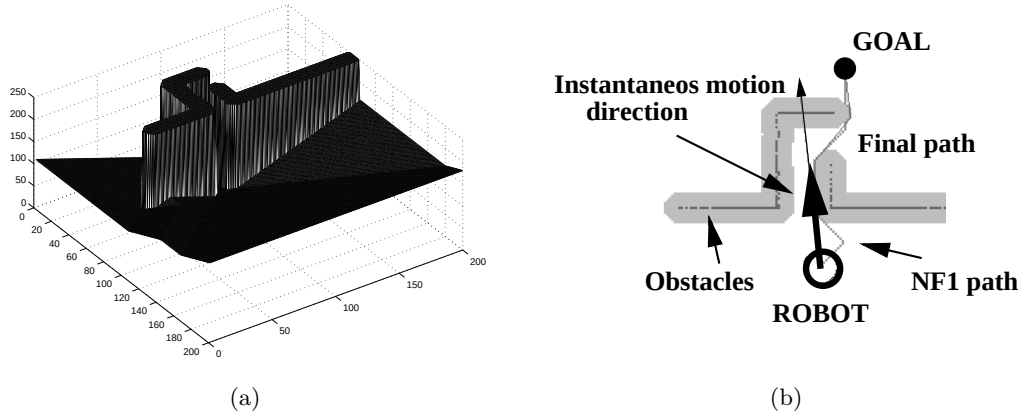
Finally, the *mapping module* and the *reactive navigation module* must cooperate to accomplish the navigation task. The *reactive navigation module* uses the obstacle information of the *mapping module*, instead of using directly the laser scan information. The obstacle information is obtained from the *occupied* cells in the occupancy-grid.

Experimental Results

Both modules have been tested on a *Nomadic XR4000* equiped with a 2D Sick laser at *LAAS-CNRS*, France (see Appendix E.0.4).

For experimentation, the grid dimensions were 10 by 10 meters, and the cell dimensions were 5 by 5 centimeters. The grid had 200 by 200 cells. Fig. 6.3b depicts the grid size. The *mapping module* took around $100msec$ to update the grid with a laser scan (361 points), and to shift the grid when necessary. The *reactive navigation module* (ND) took less than $50msec$. Both modules together gave a cycle-time around $150msec$, that was well suited for real-time collision avoidance.

An example to highlight the relevance of the framework in a dynamic environment is next presented. Fig. 6.3a depicts a real experiment where a human walked between the robot and an open passage. The goal location was at the end of the passage. Firstly, the ND drove robot towards the center of the passage, Fig. 6.3b. The human appeared in the scene, Fig. 6.3c,d. Then, the human entered in the ND security zone. Thus, the robot started an avoidance maneuver while moving towards the passage, Fig. 6.3e. Next, the human completely blocked the passage, but the robot continued the avoidance, Fig. 6.3f. In Fig. 6.3g, the human had crossed the passage, that appeared open for the robot to enter. The robot turned towards the passage while continuing



(a) (b)

Figure 6.4: a) NF1 function. b) Instantaneous path direction.

the human avoidance. Finally, the human left the ND security zone. The robot recovered the motion towards the center of the passage, Fig. 6.3h.

In this experiment, the **environment dynamism** had to be rapidly reflected into the grid model because:

1. The reactive method would not avoid the human, if it was not rapidly integrated into the grid-model.

2. The passage would remain closed after the human crossed it, if the last of the robot perceptions of the human were not eliminated from the model. Thus, the reactive method would avoid the free space.

The cooperation between the *mapping* and the *reactive navigation modules* complies with the first, second, and third requirements of the navigation system. However, the trap situations and the cyclic behaviors still need to be avoided.

6.4.3 Planning Module

The *planning module* is presented next to comply with the fourth requirement stated in Section 6.2: **trap situations and cyclic behaviors avoidance**.

This module is designed for a circular and holonomic robot working in a two dimensional space. The configuration space is $\mathbb{R}^2$. Other designs might be required for other type of robots.

The *planning module* computes two pieces of information:

1. A path connecting the current robot and goal configurations (if it exists).

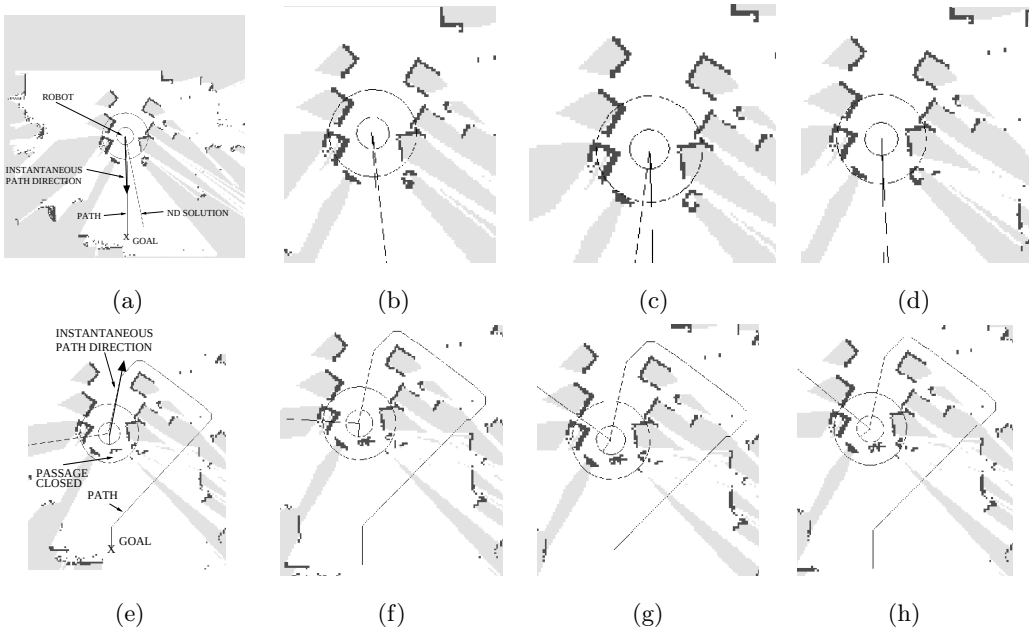


Figure 6.5: Experiment using the *Mapping*, *planning*, and *reactive navigation md-dules.*

2. The *instantaneous path direction* in order to reach the goal configuration. This direction is the main direction of the first part of the path.

The next procedure is followed to compute the path: (1) the obstacle information in the occupancy-grid is used to compute the $\mathcal{C}$-Obstacles (by enlarging the obstacles with the robot radius), see Fig. 6.4b. (2) The *Navigation Function 1*[1] [35] is constructed over the grid[2] (see Fig. 6.4a). (3) A collision-free path, connecting the current robot configuration and the goal configuration, is obtained by a gradient-search technique over the NF1. (4) The path is "stretched" with a recursive algorithm to be optimized, see Fig. 6.4b. This "stretch" strategy avoids the border effects of the NF1 function, and it modifies paths pointing towards or skirting the obstacles. The reasons for selection of this path planning algorithm are: (1) the NF1 is a minima-free naviga-

[1]The Navigation Function 1 (NF1) is a potential built over a grid representation of the configuration space. A wave is propagated among the free cells from the goal cell to all the free cells in the grid. The free cells are labeled with the minimum number of cells crossed to reach the goal location.

[2]The *free space assumption* is considered here. That is, the unknown space is considered free for the NF1 function computation.

tion function. (2) The NF1 is a grid-based navigation function, compatible with the grid-model available. (3) The path planning algorithm is simple and efficient, then it can be used at each sampling time.

The assumption here is that a path exists. When a path is not available this module produces a failure. This will be supervised by the complete system presented in the next Section.

The three modules work together as follows. The *mapping module* constructs the occupancy-grid. The *planning module* uses the occupancy-grid to compute the *instantaneous path direction*. The *reactive navigation method* uses the obstacle information in the occupancy-grid, and the *instantaneous path direction*, to compute the motion commands.

Experimental Results

For experimentation, the same robot and settings presented in Subsection 6.4.2 were used. The first meter of the path was used to compute the *instantaneous path direction*. The *planning module* imposed an additional time penalty of $100msec$. The cycle time then was $250msec$, enough for real-time.

An experiment to highlight the relevance of this framework in trap situations is presented next. Fig. 6.5a depict the initial state, that match up with Fig. 6.8e. Figs. 6.5b,c,d illustrates the robot navigating along a passage in order to reach the final location. While the robot was traveling, a human blocked the end, see Fig. 6.8f. The environmental structure was modified, and a big U-shape obstacle was created. This produced a trap situation for the robot, see Fig. 6.5e. Rapidly, the *mapping module* reflected the change, and the *planning module* calculated a new path. The *instantaneous path direction* pointed away from the U-shape configuration, see Fig. 6.5e. The ND computed the motion commands to follow this direction, see Figs. 6.5f,g,h. Therefore, the robot was driven out of the U-shape obstacle. The trap situation was avoided, see Fig. 6.8g.

The *reactive method*, *mapping module*, and *planning module* working together avoid trap situations and cyclic behaviors:

- There are no obstacle configurations that produce **trap situations** (when a solution exists within the grid-model). The *instantaneous path direction* is used to get the robot out of these situations.

- Any symmetry of the environment does not produce **cyclic behaviors** in the robot motion. The possible symmetrical motions are discriminated by the *instantaneous path direction*.

This framework complies with the first four requirements. The complete sensor-based navigation system is presented next.

6.5 The Sensor-Based Navigation System

This Section addresses the **integration of the functionalities**, the fifth requirement. An architecture is proposed that specifies the interactions among the modules. Moreover, failures in the *planning module* and in the *reactive method* are dealt with.

Planning module failure. There exist two cases where a planning algorithm cannot find a path, connecting the robot and goal configurations:

1. The goal configuration is not in free space $\mathcal{C}_{free}$ [35] (final configuration in collision with an obstacle). This is a typical situation in unknown scenarios, where goals are randomly placed for exploration. While the scenario is progressively perceived, the goal can be within an obstacle. In dynamic scenarios an object can move, or even stop, over the goal location. Even in static and known environments, this situation appears when the goal moves within an obstacle due to the robot drift.

2. The robot or the goal are surrounded by an obstacle.

Notice that both situations can appear irrespectively of the distance to the goal location. These situations could be avoided by replacing the goal location. However, the role of the navigation system is not to modify the goal location (imposed by an external agent). The consequences can drastically determine the success of the global task.

The *planning module* produces a failure in these situations. However, the system must be able to continue the navigation task (close the motion control loop).

Reactive navigation failure. The ND method uses an internal piece of information named *free walking area* (in short, it represents a region to drive the robot). If there is not *free walking area* available, the goal cannot be reached. This situation appears when the robot is completely surrounded by obstacles. The *reactive method* produces a failure in this situation.

The following architecture specifies the module interaction and deals with module failures, see Fig. 6.6. The navigation system works as follows:

Procedure Navigation System step (n, goal)
$Data_{laser}$ = *Read Laser*

$Data_{odom}$ = *Read Odometry*
$Grid_n$ = *Mapping Module(*$Data_{laser}$, $Data_{odom}$, $Grid_{n-1}$*)*
$NF1_{path}$ = *Planning Module(*$Grid_n$, *goal,* $Data_{odom}$*)*
if $NF1_{path}$ available then
  exec *Reactive Module (*$Grid_n$, $NF1_{path}$, $Data_{odom}$*)*,
else
  exec *Reactive Module (*$Grid_n$, *goal,* $Data_{odom}$*).*

Procedure Reactive Module (Obstacles, goal, $Data_{odom}$)
  if $\exists$ ($ND_{free\ walking\ area}$) then
    $(v = v_{ND}, w = w_{ND}) = ND(Obstacles, goal, Data_{odom})$
  else
    $(v = 0, w = w_{cte})$.
  $(v, w) \rightarrow Controllers$

First, the *mapping module* updates the cells of the grid and the grid location. Next, the *planning module* computes the path. If a path exists, the *reactive navigation module* computes the motion commands to direct the robot towards the *instantaneous path direction*. If not, the *reactive navigation module* drives the robot towards the goal location (*planning module* failure). Then, if there is *free walking area* available, the motion commands computed by the *reactive method* are used. If not, the motion commands stop and rotates the robot over its center (*reactive method* failure). This strategy updates the grid in all directions. This behavior is maintained until the environment changes, and a *free walking area* can be selected for motion. Otherwise, a time-out stops the robot and fires a global trap flag.

The navigation system **integrates the information** and **reacts to dynamic environments** due to the *mapping module*. The **trap situations** and **cyclic behaviors** are avoided using information of the *planning module*. The *reactive navigation method* achieves **robust navigation**. The integration is carried out within an architecture that **integrates the functionalities**. Seen as a whole, the sensor-based navigation system complies with all the requirements mentioned in the previous Sections.

6.6 Experimental Results

This Section presents the experimental results. The experiments were designed to validate the compliance of the sensor-based navigation system with the requirements of this work.

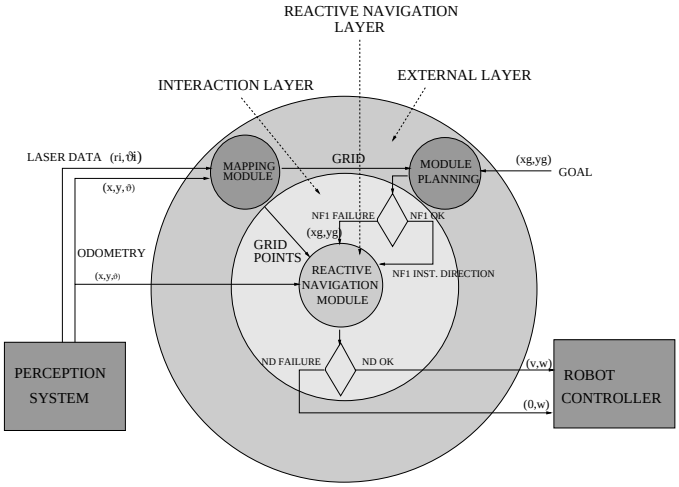


Figure 6.6: The complete navigation system scheme.

For each experiment, only the executed robot path is outlined (showing all the laser points perceived would result in a blurred Figure), see Fig. 6.7a. The robot velocity profiles are also illustrated, see Fig. 6.7b. Some snapshots have been selected for better understanding of the robot motion (notice that the floor tiles are 10cm square, and the robot diameter is 70cm), see Fig. 6.7e. Moreover, the robot location, a fraction of the grid-map, the computed path, and the ND method direction solution are shown at some selected times, see Fig. 6.7k.

The same platform and settings presented in Subsection 6.4.2 were used for experimentation. In all the experiments the environment was unknown. The scenario might be unstructured, dynamic, and non-predictable. Only the goal location was available in advance. These circumstances justify the usage of a sensor-based navigation method to move the robot.

Experiment 1: This experiment was designed to: (1) show **robust navigation** in dense, complex, and cluttered scenarios, and (2) validate the **information integration**. In order to reach the goal location, the robot had to navigate along a passage with a reduced space to maneuver, see Fig. 6.7c.

The *reactive method* successfully navigated among very close obstacles (with less than $10cm$ on both sides of the robot), see Fig. 6.7d-k,e-l,h-m,i-n. A smooth path was made by the robot, and no oscillatory behaviors were observed, see Fig. 6.7a. The selection of the ND method is justified when the robot is required to move in these scenarios, which remain troublesome for

other existing reactive methods. A deep discussion about navigation in these scenarios is presented in [44].

The *mapping module* also played an important role because:

- New sensory perceptions were rapidly integrated into the grid. Thus, the robot avoided the new obstacles when they were perceived, see the sequence of Figs. 6.7k,l,m,n.

- Past sensory perceptions remained integrated into the grid. Due to the sensor visibility constraints and its location within the robot[3], sometimes the closest obstacles were not perceived. Figs. 6.7g,h-m,i-n depict moments when this happened. However, these obstacles were perceived some time before, thus, they remained in the grid and collisions were avoided.

The velocity profile is illustrated in Fig. 6.7b. The robot moved at low speeds due to the complexity of the scenario. The maximum speed was achieved when the robot was out of the passage. The time of the experiment was $127sec$, and the average translational velocity was $0.204\frac{m}{sec}$.

Experiment 2: This experiment was designed to test the **trap situations avoidance**. Fig. 6.8c shows the initial state of the robot and the environment. The robot must navigate along the passage to reach the goal location. Three consecutive trap situations were produced by changes in the environment structure.

The robot started by navigating along the passage, Fig. 6.8d-k. While the robot was moving forward, a passage on the rigth-hand was opened (the robot could not perceive it), see Fig. 6.8d,e. Reaching the end of the passage, a human placed a box closing the main way, see Fig. 6.8f-l. The robot was trapped within a big U-shape obstacle. Due to the information provided by the *planning module*, computed from the environmental model kept by the *mapping module*, the robot rapidly turned and move backwards in the passage to avoid the trap situation. This part of the experiment is displayed step by step in Fig. 6.5. The robot perceived the open passage on the right-hand while navigating to get out of the passage. The new passage was rapidly integrated into the grid. Then, the robot moved towards the new passage, see Fig. 6.8g. Then, the human closed the new passage, see Fig. 6.8h-m. The change was rapidly modeled, and the robot was driven out of the passage, avoiding the new trap situation, see Fig. 6.8i-n. Finally, the global **trap situation was**

[3]Notice that the sensor is placed $0.24cm$ away from the center of the robot. The sensor has a 180° field of visibility.

avoided, see Fig. 6.8j, and the robot resumed the motion towards the goal location.

The path made by the robot is shown in Fig. 6.8a. The velocity profiles are illustrated Fig. 6.8b (only $150sec$ due to a memory limitation). The complete time of the task was $200sec$, and the average translational velocity was $0.233\frac{m}{sec}$.

Experiment 3: This experiment shows the robot navigating in a typical populated and changing scenario. The sensor-based navigation system must comply with all the requirements proposed in order to successfully navigate in this scenario. Fig. 6.9c depicts the initial state of the robot and the environment. The robot had to cross the hall to reach the final location.

During the first part of the experiment humans were randomly walking, building, and modifying the environment to hinder the robot motion, see Fig. 6.9c,d,e,f. In this part of the experiment the *mapping module* was rapidly reflecting the environmental changes. Then, the *reactive method* was successfully avoiding collisions. Notice that, if the sensory information were not properly integrated, a barrier of obstacles would appear in the grid. Thus, the robot would not be able to avoid the collisions, or would avoid the free space. Subsequently, the humans built an obstacle configuration that trapped the robot, see Fig. 6.9g. The *planning module* rapidly provided information used by the *reactive method* to drive the robot out to the U-shape obstacle, see Fig. 6.9h. Finally, the robot resumed the motion and reached the goal location while reacting to the human motion, see Fig. 6.9i,j.

Unfortunately the information of the experiment was lost. However, a very similar run is discussed. The highly dynamic nature of the environment is depicted in Fig. 6.9k,l. They could be compared with the Fig. 6.9c,d,e,f. Subsequently, the humans produced a trap situation, see Fig. 6.9m, similar to the Fig. 6.9g. The robot turned to the right to avoid the trap situation. Finally, the robot continued the motion in an environment continuously changing, see Fig. 6.9n.

The path executed by the robot is shown in Fig. 6.9a. The velocity profile is illustrated Fig. 6.9b (only $150sec$ due to a memory limitation). The complete time of the experiment was $170sec$, and the average translational velocity was $0.196\frac{m}{sec}$.

6.7 Discussion

The strength of the navigation system is the quality and robustness of the navigation performance. The robot successfully navigates in troublesome sce-

narios due to the compliance with the requirements presented in Section 6.2. These scenarios would produce difficulties to other existing methods, since none of them address all these requirements presented.

The limitation of the proposed sensor-based navigation system is the computational load (in our current implementation a *Pentium II* is used. To work on systems with low computational capabilities, the size of the grid map would be reduced. This would affect the locality of the navigation method proposed.

The functional modularity and the coordination, both give generality, portability, robustness, and efficacy to the navigation system. To validate this last statement, the navigation system was tested with some modifications on two indoor and one outdoor robot at *LAAS-CNRS*, France; on one indoor robot at the *Technical University of Lisbon*, Portugal; and on an indoor robot at the *University of Zaragoza*, Spain. The *reactive method* ND was modified to work on robots with non-circular shapes and with kinematic constraints, see [45]. The *mapping module* was re-designed to work with a tri-dimensional laser, and with ultrasounds [10]. For the outdoor robot an special module to process visual information was used [27]. In the *planning module* some changes were required to take into account the specific robot characteristics. The new modules were integrated within the architecture. Thus, the portability among platforms was straightforward.

The presented navigation system is not a complete navigation system. The system internal model represents a local fraction of the environment (setting the locality of the method). Then, the system relies on higher levels to accomplish a global navigation task: supervision, global planning, localization, and map building. To date, the proposed navigation system has been used as the sensor-based navigation system for a topological-based navigation [63]. A global topological model of the environment is built on-line, while the robot is self-located within the model. From the model, sub-goals are placed for exploration, and they are reached by means of the presented sensor-based navigation system. Currently, in the Robels system [51], the navigation system is one of the five sensory-motor functions used to perform the robot motion. Moreover, the navigation system has been successfully integrated as the low-level motion generator in the complete $G^{en}oM$ architecture [23] on the *Nomadic XR4000* at LAAS-CNRS, France. The navigation system is daily used to move the platform.

6.8 Conclusions

In this Chapter a sensor-based navigation system for collision avoidance in human-made scenarios has presented. Five navigation requirements were identified in order to navigate in these scenarios: robust navigation, information integration, reaction in dynamic environments, trap situations and cyclic behaviors avoidance, and functionalities integration. The navigation system compliance with these requirements has been the main design requirement. The consequence is a sensor-based navigation system that achieves robust navigation in dense, complex, cluttered, unknown, and highly dynamic scenarios. Besides, the trap situations and cyclic behaviors in the robot motion are avoided. This is illustrated in the experimental results. Moreover, the navigation system is validated by the robots that everyday make use of it.

The architecture that embeeds the funtionalities allows for module replacement, and for module failure and recovery. The function modularity and the architecture, both are the basis for the integration of the research, presented in previous Chapters, in a unique framework. Besides, this navigation system is the basis for the development of a complete navigation system.

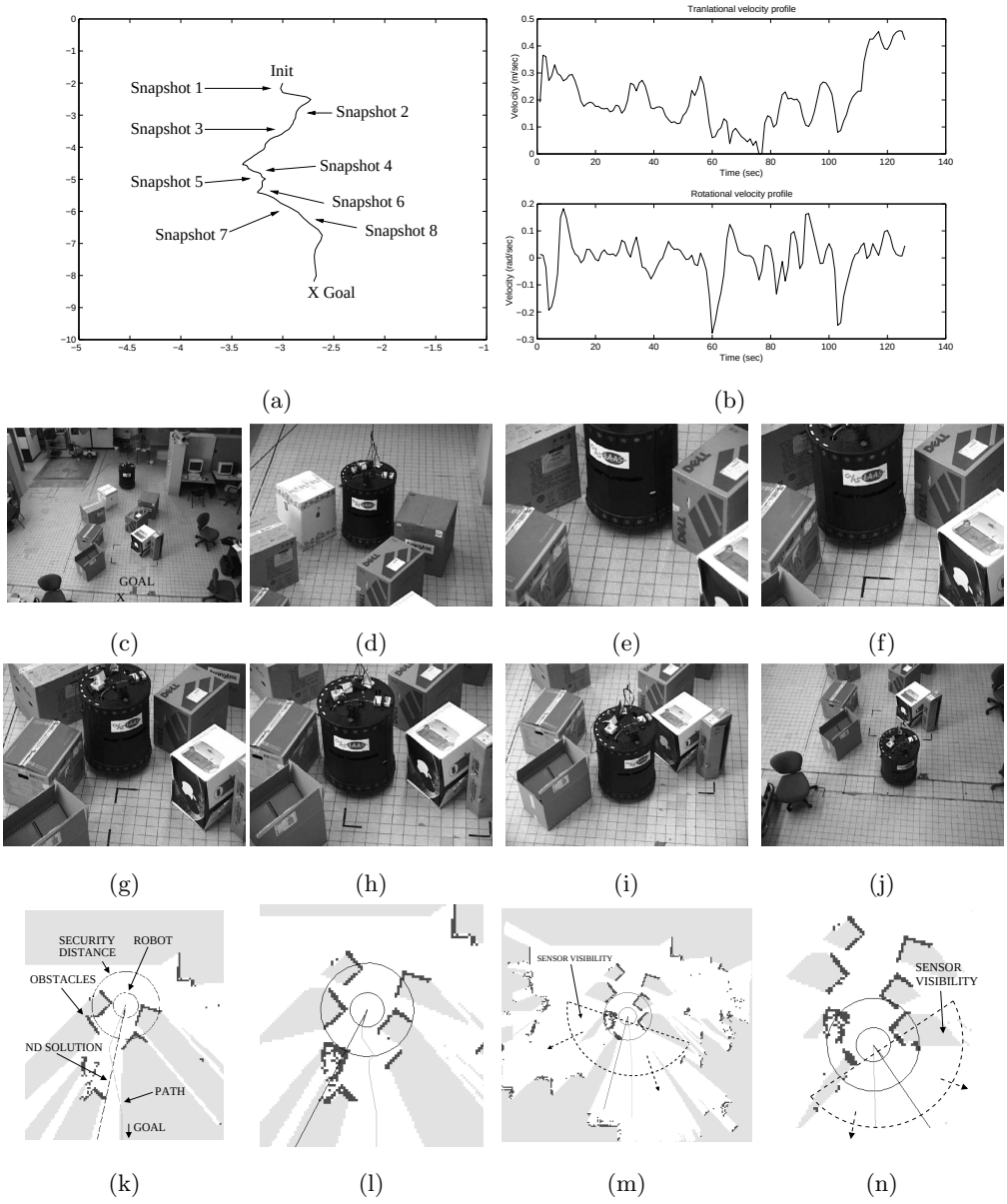


Figure 6.7: Experiment 1.

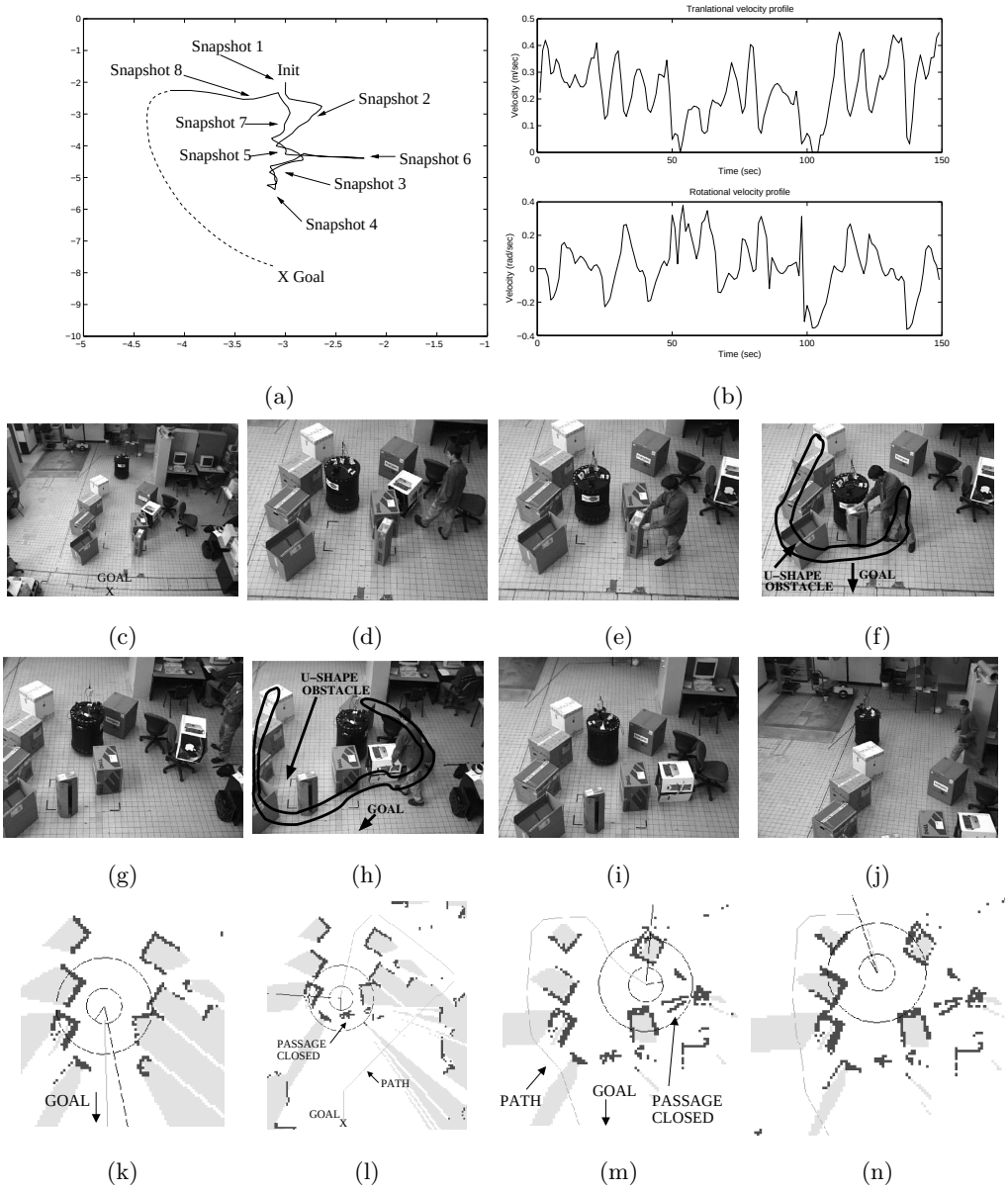


Figure 6.8: Experiment 2.

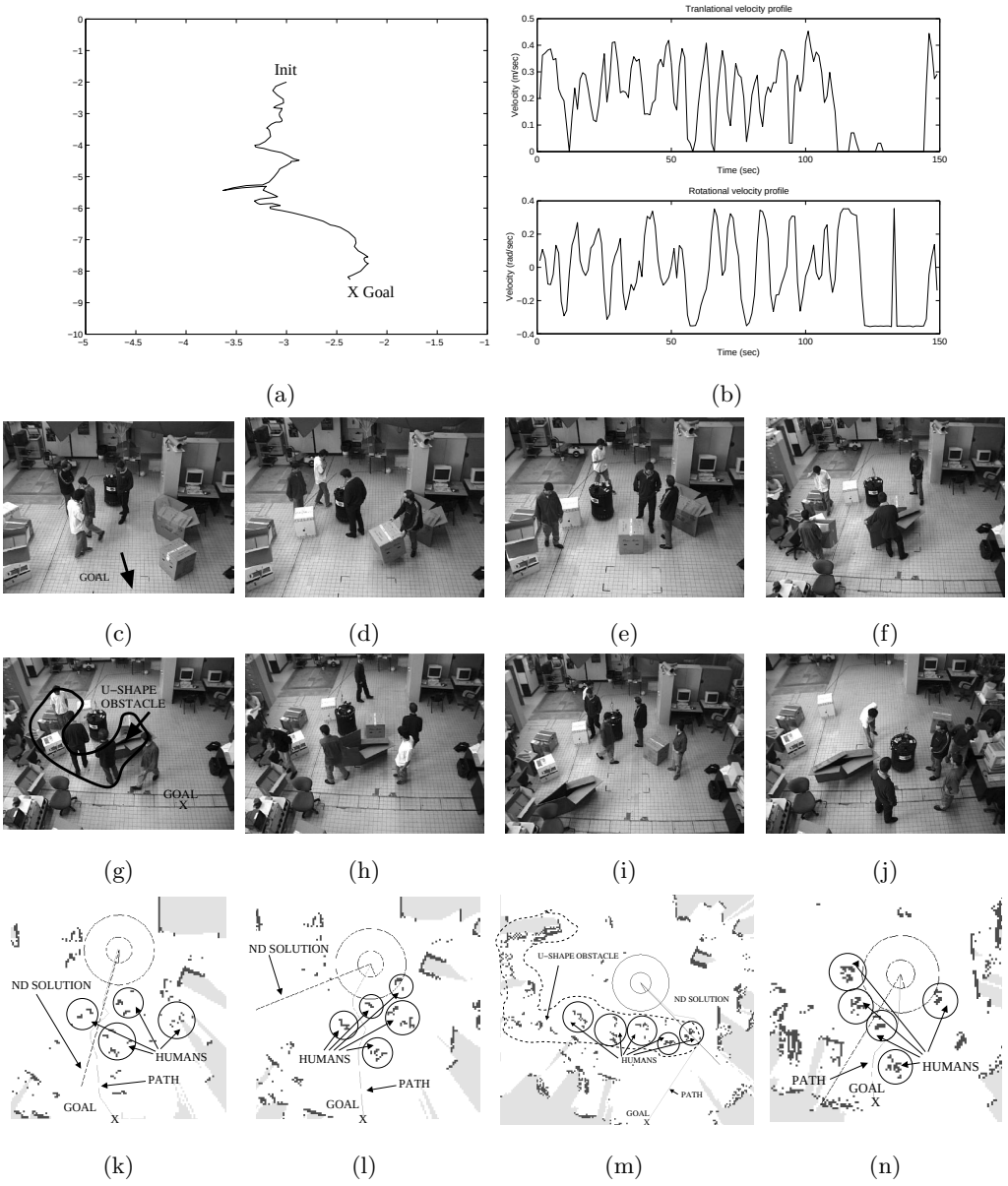


Figure 6.9: Experiment 3.

Chapter 7

Conclusions

The work presented in this memory contributes to the *sensor-based motion planning* field. Each chapter has a different nature, and addresses a different topic in this field. The main contributions are:

- **Sensor-based motion planning**

  Chapter 2 addresses the *sensor-based motion planning* by presenting the design of a reactive navigation method.

  The reactive navigation method has been designed using the *situated-activity* paradigm of behavioral design. By using this paradigm, the reactive navigation problem is broken down into sub-problems. This strategy simplifies the complete reactive navigation problem. As a consequence, a new method implementation of the proposed design leads to a new reactive navigation method. The new method might successfully move a robot in more troublesome scenarios than other methods do. That is, the robot might be able to successfully navigate in dense, complex, and cluttered environments.

  To demonstrate the utility of the design, a geometric implementation called *Nearness Diagram Navigation* has been presented. The method has been validated using a real platform. Up to date the author does not know any other method or approach that performs better navigation than the *Nearness Diagram Navigation*.

- **Under-constraint Sensor-based Motion Planning**

  The Chapter 3 presents and under-constrained solution to extend the *Nearness Diagram Navigation* method, to work on robots with non-circular shapes considering the kinematics and dynamics. By using the

proposed framework, constrained robots are successfully move in dense, complex, and cluttered scenarios.

Experimental results on five different robots using different sensors validate the utility, and easy portability, of this reactive navigation method. The author believes that this method is one of the most robust sensor-based motion planners to drive constrained robots in troublesome scenarios.

- **Sensor-based motion planning with kinematic constraints**

  To address the kinematic constraints, Chapter 4 presents the *Ego-Kinematic Space*. This space can be used to convert existing reactive navigation methods that do not address the kinematic constraints in methods that do. A space, where the methods produce solutions that comply with the kinematics, has been constructed. To validate this framework, two reactive navigation methods have been successfully used to safely drive a kinematic constrained platform among locations. Both methods do not address the kinematics. The author believes that this is a wide solution to address the kinematics in sensor-based motion planning.

  Another contribution of this Chapter is to demonstrate, that the subspace that really matters for sensor-based motion planning with kinematic constraints, is two-dimensional (instead of three-dimensional, as it has been classically addressed). The author believes that this result will alleviate significantly the computations required by new methods in the future.

- **Sensor-based motion planning with dynamic constraints**

  To address the dynamic constraints, Chapter 5 presents the *Ego-Dynamic Space*. This space is constructed by using the dynamic constraints. Thus, reactive methods used in this space compute solutions that comply with the robot dynamics constraints. To validate the framework two reactive navigation methods have been successfully used to safely drive a dynamic constrained platform among locations. Both methods do not address dynamics. The author believes that this is a wide solution to address the dynamic constraints into the sensor-based motion planning discipline.

- **Sensor-based navigation system**

  This thesis also addresses the improvement of the behavior of reactive navigation methods. Chapter 6 proposes an architecture to coordinate

some modules in order to improve the reactivity of the pure navigation base algorithm. By means of the complete framework, the robot successfully navigates in highly dynamic scenarios, while avoiding the well-known trap situations. The proposed framework has been validated with a real platform.

Moreover, this Chapter is the basis for the integration of all the technologies proposed in other Chapters in a unique framework.

To discuss the future work, consider the following two questions: (1) how many mobile vehicles exists (earth, air, water, space, ...)? (2) For which of them, safe motion is computed **robustly**? Consider shape, kinematics, dynamics, sensors difficulties, different scenarios .... There are many problems still to be solved, at least in *Sensor-Based Motion Planning*, to develop applications in the real world.

Bibliography

[1] R. Alami, I. Belousov, S. Fleury, M.Herb, F. Ingrand, J. Minguez, and B. Morisset. Diligent: Towards a human-friendly navigation system. In *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, pages 2094–2100, Takamatsu, Japan, 2000.

[2] J.C. Alvarez, A. Shkel, and V. Lumelsky. Building topological models for navigation in large scale environments. In *IEEE International Conference on Robotics and Automation*, Leuven, Belgium, 1998.

[3] R.C. Arkin. *Behavior-Based Robotics*. The MIT Press, 1999.

[4] K. Arras, J. Persson, N. Tomatis, and R. Siegwart. Real-time Obstacle Avoidance for Polygonal Robots with a Reduced Dynamic Window. In *IEEE Int. Conf. on Robotics and Automation*, pages 3050–3055, Washington, USA, 2002.

[5] J. Asensio and L. Montano. A Kinematic and Dynamic Model-Based Motion Controller for Mobile Robots. In *15th IFAC World Congress*, Barcelona, Spain, 2002.

[6] K. Azarm and G. Schmidt. Integrated mobile robot motion planning and execution in changing indoor environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 298–305, Munchen, Germany, 1994.

[7] J. Barraquand and J.C. Latombe. On nonholonomic mobile robots and optimal maneuvering. In *Intelligent Symposium on Intelligent Control*, pages 340–346, Albany, 1989.

[8] J.D. Boissonnat and X.N. Xi. Accesibility region for a car that only move forward along optimal paths. In *Research Report INRIA 2181*, Sophia-Antipolis, France, 1994.

[9] J. Borenstein and Y. Koren. Real-Time Obstacle Avoidance for Fast Mobile Robots. *IEEE Transactions on Systems, Man and Cybernetics*, 19(5):1179–1187, 1989.

[10] J. Borenstein and Y. Koren. Histogramic in-Motion Mapping for Mobile Robot Obstacle Avoidance. *IEEE Journal on Robotics and Automation*, 7(4):535–539, 1991.

[11] J. Borenstein and Y. Koren. The Vector Field Histogram–Fast Obstacle Avoidance for Mobile Robots. *IEEE Transactions on Robotics and Automation*, 7:278–288, 1991.

[12] O. Brock. *Generating Robot Motion: The Integration of Planning and Execution.* PhD thesis, Stanford University, 1999.

[13] O. Brock and O. Khatib. High-Speed Navigation Using the Global Dynamic Window Approach. In *IEEE Int. Conf. on Robotics and Automation*, pages 341–346, Detroit, MI, 1999.

[14] O. Brock and O. Khatib. Real-Time Replanning in High-Dimensional Configuration Spaces using Sets of Homotopic Paths. In *IEEE Int. Conf. on Robotics and Automation*, pages 550–555, San Francisco, USA, 2000.

[15] Joachim M. Buhmann, Wolfram Burgard, Armin B. Cremers, Dieter Fox, Thomas Hofmann, Frank E. Schneider, Jiannis Strikos, and Sebastian Thrun. The mobile robot RHINO. *AI Magazine*, 16(2):31–38, 1995.

[16] A. Calcitti and R. Zapata. Reactive Behaviours of Mobile Manipulators Based on the DWZ Approach. In *IEEE Int. Conf. on Robotics and Automation*, Korea, 2001.

[17] J. A. Castellanos, J.M.M. Montiel, J. Neira, and J. D. Tardós. The SPmap: A probabilistic framework for simultaneous localization and map building. *IEEE Trans. Robotics and Automation*, 15(5):948–952, 1999.

[18] J. A. Castellanos and J. D. Tardós. *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach.* Kluwer Academic Publishers, Boston, 1999.

[19] L. Chenqing, M. Ang, H. Krishman, and L. Yong. Virtual Obstacle Concept for Local-minimum-recovery in Potential-field Based Navigation. In *IEEE Int. Conf. on Robotics and Automation*, pages 983–989, San Francisco, USA, 2000.

[20] M. W. M. G. Dissanayake, P. Newman, H. F. Durrant-Whyte, S. Clark, and M. Csorba. A solution to the simultaneous localization and map building (slam) problem. *IEEE Trans. Robotics and Automation*, 17(3):229–241, 2001.

[21] L.E Dubins. On curves of minimal lenght with a constraint on average curvature and with prescribed initial and terminal positions and tangents. *Amer. J. Math.*, 57:497–516, 1957.

[22] W. Feiten, R. Bauer, and G. Lawitzky. Robust Obstacle Avoidance in Unknown and Cramped Environments. In *IEEE Int. Conf. on Robotics and Automation*, pages 2412–2417, San Diego, USA, 1994.

[23] S. Fleury. *Architecture de contrôle distribueé pour robots autonomes: principes, conception et applications.* PhD thesis, Université Paul Sabatier, 1996.

[24] J. Foley, A. Van Dam, S. Feiner, and J. Hughes. *Computer Graphics, principles and practice.* Addison Wesley edition 2nd, 1990.

[25] D. Fox, W. Burgard, and S. Thrun. The Dynamic Window Approach to Collision Avoidance. *IEEE Robotics and Automation Magazine*, 4(1), 1997.

[26] M. Ginger. Universal planning: An (almost) universally bad idea. *AI Magazine*, 10(4):40–44.

[27] H. Haddad, M. Khatib, S. Lacroix, and R. Chatila. Reactive Navigation in Outdoor Environments Using Potential Fields. In *IEEE Int. Conf. on Robotics and Automation*, volume 2, pages 1232–1237, Leuven, Belgium, 1998.

[28] M. Hebert, C. Thorpe, and A. Stentz. *Intelligent Unmanned Ground Vehicles: Autonomous Navigation Research at Carnegie Mellon.* Kluwer Academic Publishers, 1997.

[29] M. Khatib. *Sensor-based motion control for mobile robots.* PhD thesis, LAAS-CNRS, 1996.

[30] O. Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *Int. Journal of Robotics Research*, 5:90–98, 1986.

[31] O. Khatib and et al. Robotics and interactive simulation. *Communications of the ACM*, 2002.

[32] S. Koenig and R. Simmons. Xavier: A robot navigation architecture based on partially observable markov decision process models. In R. Bonasso D. Kortenkamp and R. Murphy, editors, *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, pages 91 – 122. MIT Press, 1998.

[33] Y. Koren and J. Borenstein. Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation. In *IEEE Int. Conf. on Robotics and Automation*, volume 2, pages 1398–1404, Sacramento, CA, 1991.

[34] B. H. Krogh and C. E. Thorpe. Integrated Path Planning and Dynamic Steering control for Autonomous Vehicles. In *IEEE Int. Conf. on Robotics and Automation*, pages 1664–1669, San Francisco, USA, 1986.

[35] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic, 1991.

[36] J.P. Laumond, S. Sekhavat, and F. Lamiraux. Guidelines in nonholonomic motion planning for mobile robots. *Robot Motion Planning and Control*, 229, 1998.

[37] J.P. Laumond, M. Taix, and P. Jacobs. A motion planner for car-like robots based on a mixed global/local approach. In *IEEE International Workshop on Intelligent Robots and Systems*, pages 765–773, Japan, 1990.

[38] J. J. Leonard and H. J. S. Feder. A computationally efficient method for large-scale concurrent mapping and localization. In D. Koditschek and J. Hollerbach, editors, *Robotics Research: The Ninth International Symposium*, pages 169–176. Springer Verlag, Snowbird, Utah, 2000.

[39] T. Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, 32(2):108–120, 1983.

[40] V. Lumelsky and A. Shkel. Incorporating body dynamics into the sensor-based motion planning paradigm. the maximum turn strategy. In *IEEE International Conference on Robotics and Automation*, Nagoya, Japan, 1995.

[41] C. Marques. Multi-sensor navigation for soccer robots. Master's thesis, Instituto Superior Tecnico, Portugal, 2001.

[42] C. Marques and P. Lima. *Multi-sensor Navigation for Soccer Robots*. Springer-Verlag, Berlin, Grmany, 2002.

[43] A. Masoud, S. Masoud, and M. Bayoumi. Robot navigation using a pressure generated mechanical stress field, the biharmonical potential approach. In *IEEE International Conference on Robotics and Automation*, pages 124–129, San Diego, USA, 1994.

[44] J. Minguez and L. Montano. Nearness Diagram Navigation (ND): A New Real-Time Collision Avoidance Approach. In *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, pages 2094–2100, Takamatsu, Japan, 2000.

[45] J. Minguez and L. Montano. Robot Navigation in Very Complex Dense and Cluttered Indoor/Outdoor Environments. In *15th IFAC World Congress*, Barcelona, Spain, 2002.

[46] J. Minguez, L. Montano, and O. Khatib. Reactive Collision Avoidance for Navigation at High Speeds or Systems with Slow Dynamics. In *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, pages 588–594, Lausanne, Switzerland, 2002.

[47] J. Minguez, L. Montano, and J. Santos-Victor. Reactive Collision Avoidance for Non-holonomic Robots using the Ego-Kinematic Space. In *IEEE Int. Conf. on Robotics and Automation*, pages 3074–3080, Washington, USA, 2002.

[48] J. Minguez, L. Montano, N. Simeon, and R. Alami. Global Nearness Diagram Navigation (GND). In *IEEE International Conf. on Robotics and Automation*, pages 33–39, Seoul, Korea, 2001.

[49] B. Mirtich and J. Canny. Using skeletons for nonholonomic path planning among obstacles. In *IEEE Int.Conf.on Robotics & Automation*, May 1992.

[50] L. Montano and J. Asensio. Real-Time Robot Navigation in Unstructured Environments Using a 3D Laser Rangefinder. In *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, volume 2, pages 526–532, Grenoble, France, 1997.

[51] B. Morisset and M. Gallab. Learning how to combine sensory-motor modalities for a robust behavior. *Advances in Plan-Based Control of Robotic Agents, Lecture Notes in Artificial Intelligence 2466, Springer*, pages 157–178, 2002.

[52] J. Persson. Obstacle avoidance for mobile robots. Master's thesis, EPFL, 2000.

[53] S. Quinlan. *Real-Time Modification of Collision Free Paths*. PhD thesis, Stanford University, December, 1994.

[54] S. Quinlan and O. Khatib. Elastic Bands: Connecting Path Planning and Control. In *IEEE Int. Conf. on Robotics and Automation*, volume 2, pages 802–807, Atlanta, USA, 1993.

[55] J.A. Reeds and L.A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 145, 1990.

[56] R. Simmons. The Curvature-Velocity Method for Local Obstacle Avoidance. In *IEEE Int. Conf. on Robotics and Automation*, pages 3375–3382, Minneapolis, USA, 1996.

[57] S. Thrun, W. Burgard, and D. Fox. A real-time algorithm for robot mapping with applications to multirobot and 3d mapping. In *IEEE Int. Conf. on Robotics and Automation*, pages 321–328, San Francisco, CA, 2000.

[58] R. B. Tilove. Local Obstacle Avoidance for Mobile Robots Based on the Method of Artificial Potentials. In *IEEE Int. Conf. on Robotics and Automation*, volume 2, pages 566–571, Cincinatti, OH, 1990.

[59] I. Ulrich and J. Borenstein. VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots. In *IEEE Int. Conf. on Robotics and Automation*, pages 1572–1577, 1998.

[60] I. Ulrich and J. Borenstein. VFH*: Local Obstacle Avoidance with Look-Ahead Verification. In *IEEE Int. Conf. on Robotics and Automation*, pages 2505–2511, San Francisco, USA, 2000.

[61] M. Vendittelli, J.P. Laumond, and C. Nissoux. Obstacle distance for car-like robots. *IEEE Transactions on Robotics and Automation*, 15, 1999.

[62] T. Wilkman, M. Branicky, and W. Newman. Reflexive Collision Avoidance: A Generalized Approach. In *IEEE Int. Conf. on Robotics and Automation*, Atlanta, USA, 1993.

[63] D. Van Zwynsvorde, T. Simeon, and R. Alami. Building topological models for navigation in large scale environments. In *IEEE Int. Conf. on Robotics and Automation*, pages 23–29, Seul, Korea, 2001.

Appendix A

A "Navigable" *Region*

This Appendix introduces a local algorithm that verifies if a *region* is "navigable" for a circular and holonomic robot. Firstly, a local algorithm is presented to check whether the robot can reach a location in the space. Next, the use of the algorithm to verify if a *region* is "navigable" is discussed.

A.0.1 The basic algorithm

The **inputs** of the algorithm are:

1. The robot location, $\mathbf{x_r}$, and robot radius R.
2. A location in the space, $\mathbf{x_p}$.
3. A list of N obstacle points, L. An obstacle point is $\mathbf{x_j^L}$

The **output** of the algorithm is, if the point $\mathbf{x_p}$ *can be reached* from the robot location $\mathbf{x_r}$.

The algorithm computes the existence of a path that connects the robot, $\mathbf{x_r}$, and a location, $\mathbf{x_p}$. The algorithm does not compute a path.

The plane is divided in four semi-planes by: (1) the line that contains $\mathbf{x_r}$ and $\mathbf{x_p}$, and (2) the perpendicular line to the previous line over $\mathbf{x_r}$. The space is divided in four semi-planes FL, FR, BL, BR[1], see Fig. A.1a. Then:

1. Compute the distances $d_i(\mathbf{x_p}, \mathbf{x_i^L})$. If $\exists i / d_i(\mathbf{x_p}, \mathbf{x_i^L}) < R$: $\mathbf{x_p}$ *cannot be reached*. If not,
2. Compute the distances $d_{j,k}(\mathbf{x_j^L}, \mathbf{x_k^L})$, $\mathbf{x_j^L} \in FR$ and $\mathbf{x_k^L} \in FL$. If $\forall j, k, d_{j,k}(\mathbf{x_j^L}, \mathbf{x_k^L}) > 2.R$: $\mathbf{x_p}$ *can be reached*. If not,

[1] F: forward. B: backward. R: right. L: left.

3. For those points $\mathbf{x_l^L}$ that do not verify the previous statement, compute the distances $d_l(\mathbf{x_l^L}, \mathbf{x_r})$. If $\forall l, d_l(\mathbf{x_l^L}, \mathbf{x_r}) > d(\mathbf{x_p}, \mathbf{x_r})$: $\mathbf{x_p}$ *can be reached*. If not, $\mathbf{x_p}$ *cannot be reached*.

This is the basic algorithm, but some computations are not required. The algorithm is next discussed:

1. The algorithm checks whether the point $\mathbf{x_p}$ is within any $\mathcal{C}$-Obstacle[2] [39]. Thus, the point could not be reached.

2. The algorithm checks intersections among $\mathcal{C}$-Obstacles in different semi-planes, FR and FL. If there are not intersections, a non-empty set of paths that joins the $\mathbf{x_r}$ and $\mathbf{x_p}$ exists, see Fig. A.1a,b. Many collision-free paths exist within this set.

3. If there are intersections among $\mathcal{C}$-Obstacles of different semi-planes, the distance to the closest $\mathcal{C}$-Obstacle that intersects is computed. Comparing this distance with the distance to $\mathbf{x_p}$, it is verified if $\mathbf{x_p}$ belongs to this set of paths. Thus, a path exists connecting $\mathbf{x_r}$ and $\mathbf{x_p}$.

A.0.2 Implications with reactive navigation

This algorithm is useful for reactive navigation. The algorithm computes if the robot can reach a given point of the space, without explicitly computing any path. Then, the role of the reactive navigation is to drive the robot towards that point (having in advance the guaranty that the point can be reached). The point could be a landmark point, or the goal location itself. Figs. A.1a,b illustrate an example that checks whether a point $\mathbf{x_p}$ can be reached. The solution of the algorithm is that the point can be reached.

The algorithm is local because global information is not provided. Then, the algorithm fails in some cases that require global information. This situation is illustrated in Fig. A.1c. The output of the algorithm is that $\mathbf{x_p}$ *cannot be reached*, while a solution exists turning to the left in the middle of the passage. This type of cases cannot be avoided by using local algorithms. A global information must be added to the system to solve these cases, see Section 2.7.G.

[2] A $\mathcal{C}$-Obstacle in this case is computed by enlarging each obstacle point with the robot radius.

A.0.3 The "navegability" of a *region*

The presented algorithm is used to verify the "navigability" of a *region*. There are two cases:

1. If the goal location is inside the *region*. The algorithm checks if the goal location *can be reached*.

2. If the goal location is not inside the *region*. The algorithm checks if the middle point of the *region gap*, which is the closest to the goal location, *can be reached*. The middle point of the *gap* is used as a landmark in order to reach the goal location.

Figs. A.1a,b shows how the algorithm is used to verify whether the *gap* of the *region* can be reached. The solution of the algorithm is that the point can be reached. Thus, the *region* will be "navigable", and becomes the *free walking area*.

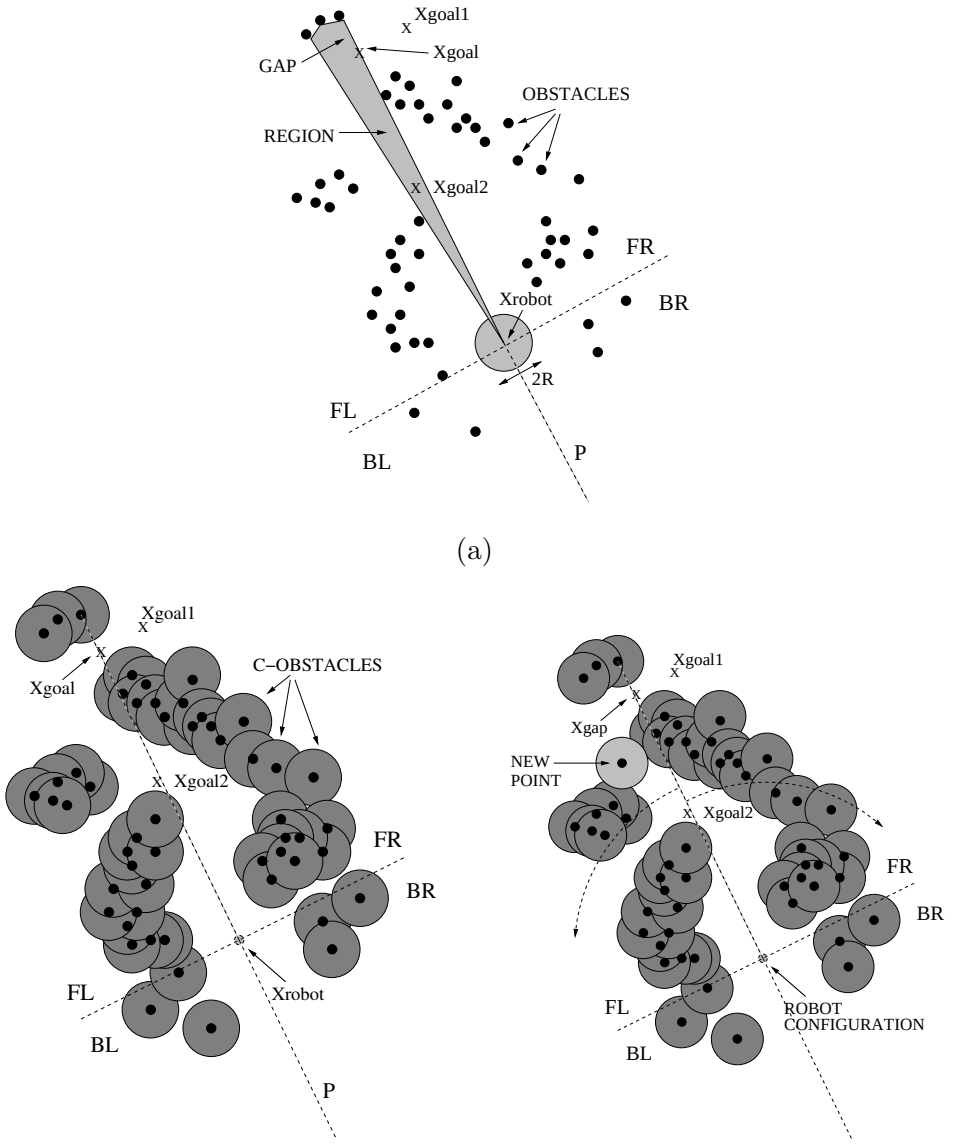


Figure A.1: Example of "navigability" checking of a region.

Appendix B

$\mathcal{C}$-Obstacle Region in R^{1mc}

The goal of this Appendix is to describe an algorithm to compute a discretization of the $\mathcal{C}$-Obstacle region, $\mathcal{R}_{obs}^{1mc}$, in the *Reachable Set of a single Motion Command*, $\mathcal{R}^{1mc}$.

The inputs of the algorithm are:

1. A list of N points of the robot boundary, (x_c^i, y_c^i).
2. A list of M obstacle points, (x_p^j, y_p^j).

The output is a list of $N \times M$ points, $(x_{sol}^{i,j}, y_{sol}^{i,j})$, of the $\mathcal{R}_{obs}^{1mc}$ boundary.

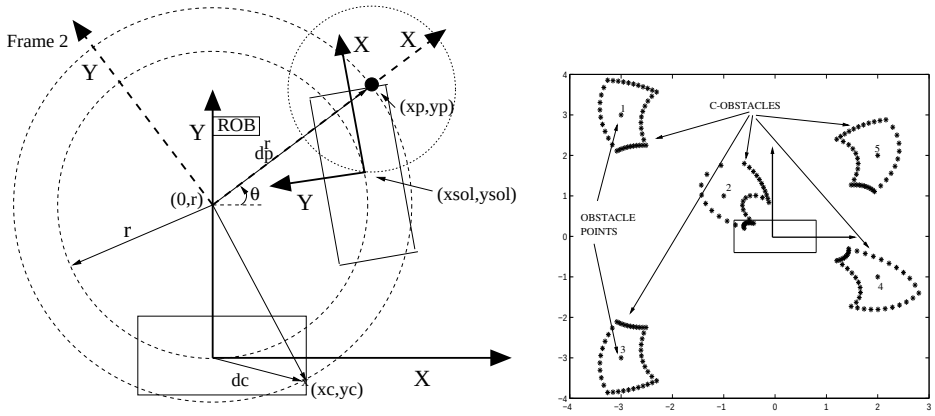
The algorithm is a brute-force method with complexity $N \times M$. However, the algorithm complies with the requirements mentioned in Section 4.3. Improvements of the algorithm remain an open problem. The current algorithm has the advantage that can be used for any robot shape (even non-polygonal shapes).

For each point (x_c^i, y_c^i) and (x_p^j, y_p^j) the following procedure is repeated to compute the $\mathcal{C}$-Obstacle bounds $(x_{sol}^{i,j}, y_{sol}^{i,j})$:

Procedure (x_{sol}, y_{sol})=Point(x_c, y_c, x_p, y_p)
The algorithm is presented for an obstacle point $(x_p, y_p) \in \mathbb{R}^2$ in the first quadrant $x_p, y_p \geq 0$. For points in other quadrants, some signs would have to be to be modified. The algorithm computes the circle that contains the location (x_{sol}, y_{sol}), and that verifies $(x_c, y_c) = (x_p, y_p)$.

The distance between the instantaneous center of rotation $(0, r)$, and the point (x_p, y_p) is d_p^r. The radius r of the circle that verified $(x_p, y_p) = (x_c, y_c)$ is computed by:

$$\begin{cases} x_c^2 + (y_c - r)^2 = (d_p^r)^2 \\ x_p^2 + (y_p - r)^2 = (d_p^r)^2 \end{cases} \quad \Rightarrow \quad r = \frac{x_c^2 + y_c^2 - (x_p^2 + y_p^2)}{y_c - y_p} \tag{B.1}$$



(a) (b)

Figure B.1: (a) Problem geometry. (b) $\mathcal{W}$ and $\mathcal{R}^{1mc}$. geometry.

The origin Frame 2 is $(0, r)$, and the y-axis is oriented towards the point (x_p, y_p). In this frame, it is verified: (1) The circle with radius r contains (x'_{sol}, y'_{sol}). (2) The circle with radius d_c contains (x'_{sol}, y'_{sol}).

$$\left\{ \begin{array}{c} (x - d_p^r)^2 + y^2 = d_c^2 \\ x^2 + y^2 = r^2 \end{array} \right. \Rightarrow \left\{ \begin{array}{l} x'_{sol} = \frac{(d_p^r)^2 + r^2 - d_c^2}{2.d_p^r} \\ y'_{sol} = -sign(x_c).\sqrt{r^2 - (x'_{sol})^2} \end{array} \right. \tag{B.2}$$

A rigid transformation is applied to transform the point (x'_{sol}, y'_{sol}) in Frame 2, in the robot frame.

$$\theta = \arcsin(\frac{y_p - r}{d_p^r}) \tag{B.3}$$

$$\left(\begin{array}{c} x_{sol} \\ y_{sol} \end{array} \right) = \left(\begin{array}{ccc} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & r \end{array} \right) \left(\begin{array}{c} x'_{sol} \\ y'_{sol} \\ 1 \end{array} \right) \tag{B.4}$$

Fig. B.1b illustrates the workspace, $\mathbb{R}^2$, and the *Reachable Set of a Single Motion Command*, $\mathcal{R}^{1mc}$, over imposed. The Figure depicts five obstacle points in $\mathcal{W}$, and the bounds of the $\mathcal{C}$-Obstacle region, $\mathcal{R}_{obs}^{1mc}$, computed by the algorithm.

Appendix C

The Selection of the Atan() Function

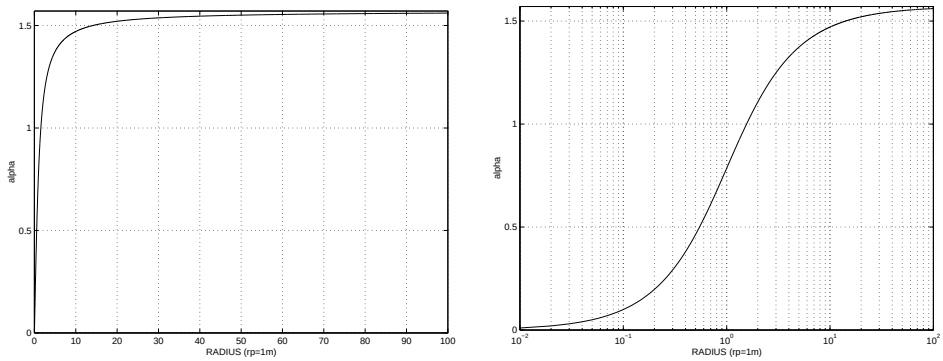
The goal of this Appendix is to justify the selection of the $\alpha = atan(\frac{R}{r_p})$. This function converts $R =]-\infty, \infty[$ into an angular descriptor $\alpha = [-\frac{\pi}{2}, \frac{\pi}{2}]$.

$r_p = \frac{d_m}{2}$, with d_m the diameter of the region to transform, see Fig. 4.3. The function $\alpha = atan(\frac{R}{r_p})$ converts into an angular index, α, the relation between the circle with radius R and the circle with radius r_p. In the following analysis the case $R > 0$ is considered.

In sensor-based motion planning, a high resolution for radius close to r_p is required, and low resolution in the other case. (e.g $r_p = 2m$ is a typical value. A turning radius $R = r_p \pm 1m$ makes a big difference in the executed motion. On the other hand, with $r_p = 100m$, a turning radius $R = 100m \pm 1m$ makes no difference, both radius produce roughly the straight motion.) This requirement is fully verified by the selected function:

1. The region to transform (circumference with radius d_m) is divided by r_p in four regions with the same area, see Fig. 4.3. The circles within $R \in [0, r_p]$ and $R \in [r_p, \infty[$, that correspond to equal-area parts within the region to transform, are mapped into equal-length intervals $\alpha \in [0, \pi/4]$ and $\alpha \in [\pi/4, \pi/2]$.

2. The resolution of the transformation is maximal for low values of R. Radius closed to r_p are represented in more detail, as opposed to areas far away from the vehicle. Fig. C.1a depicts the transformation. Fig. C.1b shows the same information, with the horizontal axis in logarithmic scale. In radius $0.1 < \frac{R}{r_p} < 10$ the angle varies in an approximate linear manner in this scale.



(a) (b)

Figure C.1: a) $\alpha = \arctan(\frac{R}{r_p})$ for $r_p = 1m$. b) $\alpha = \arctan(\frac{R}{r_p})$ for $r_p = 1m$ in logarithmic scale.

Appendix D

Error in the ED-space

This Appendix characterizes the error that arises from the EDT extension from $\mathbb{R}$ to $\mathbb{R}^2$. In the one-dimensional case, $\mathbb{R}$, the following condition is verified: $\vec{\mathbf{d}}_{\mathbf{obs}} = \vec{\mathbf{d}}_{\mathbf{eff}}$. The EDT applied to $\mathbb{R}^2$ creates a space where the previous equation is not always preserved: the particle describes a quadric (parabola) during the braking policy, see Fig. D.1. The consequence is that a direction in the ED-space does not necessarily correspond to the same direction in $\mathbb{R}^2$.

The collision checking is carried out in the ED-space following motion directions. In the ideal case, the obstacle direction in $\mathbb{R}^2$ and the obstacle direction in the ED-space have to be equal. The angle between a direction in $\mathbb{R}^2$, and the correspondent direction in the ED-space, α_{error}, is given by:

$$\alpha_{error} = atan(\frac{d_{obs_y}}{d_{obs_x}}) - atan(\frac{d_{eff_y}}{d_{eff_x}}) \tag{D.1}$$

where $\alpha_{error} = f(d_{obs_x}, d_{obs_y}, a_b, T)$.

The first quadrant of the function $\alpha_{error} = abs(f(d_{obs_x}, d_{obs_y}, 0.75\frac{m}{sec^2}, 0.4sec))$ is illustrated in Fig. D.2a. The values of a_b and T are the real ones used in the experiments performed with the *XR4000 Nomadic* platform.

The α_{error} function has the following properties:

1. $\alpha_{error}(x, y) = 0$ for the following values: $(0, y)$, $(x, 0)$, and (x, x).

2. For local obstacle avoidance, obstacles closer than $3m$ are usually taken into account. The $max(\alpha_{error}) = 9.49°$.

3. The mean of the α_{error} distribution for obstacles closer than $3m$ is $\mu(\alpha_{error}) = 5.23°$

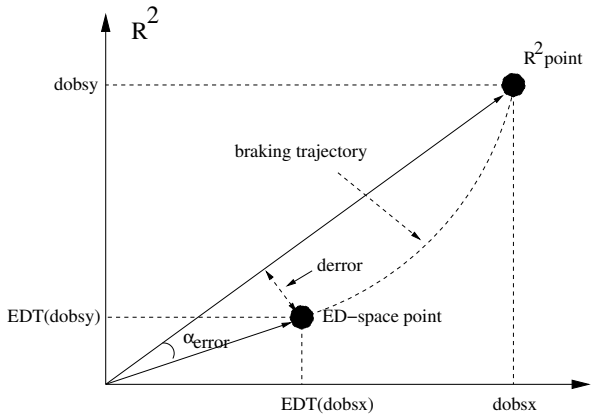


Figure D.1: ED-space error in $\mathbb{R}^2$

The error d_{error} is the distance between a location in the ED-space, and the projection in the $\mathbb{R}^2$ direction, see Fig. D.1. This distance measures the distance error between a point in the ED-space, and where the point had to be in the ideal case.

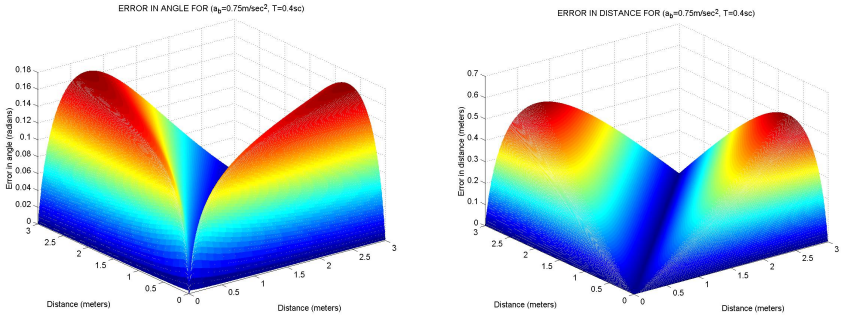
$$d_{error} = \sqrt{d_{obs_x}^2 + d_{obs_y}^2}.sin(\alpha_{error}(d_{obs_x}, d_{obs_y})) \tag{D.2}$$

The first quadrant of the function $d_{error} = abs(f(d_{obs_x}, d_{obs_y}, 0.75\frac{m}{sec^2}, 0.4sec))$ is illustrated in Fig. D.2b.

The d_{error} function has the following properties:

1. $d_{error}(x, y) = 0$ for the following values: $(0, y)$, $(x, 0)$, and (x, x).
2. For local obstacle avoidance, obstacles closer than $3m$ are usually taken into account. The $max(d_{error}) = 0.51m$.
3. The mean of the d_{error} distribution for obstacles closer than $3m$ is $\mu(d_{error}) = 0.20m$

d_{error} is an unbounded function whose value increases with the distance to an obstacle (around $3m$, $max(d_{error}) = 0.51m$). To derive a conclusion of how far this error is significant, it must be compared with the maximum distance that the robot travels in one sample period, T. Traveling this distance is when collisions can occur. Notice that a new ED-space is constructed every sample period, T. For the experiments with the real platform, the maximum velocity was set to $v_{max} = 0.8\frac{m}{sec}$. This gives a maximum travel distance of $d_{max} =$



(a) (b)

Figure D.2: a) $\alpha = f(0\ldots3, 0\ldots3, 0.75\frac{m}{sc^2}, 0.4sec)$. b) $d_{error} = f(0\ldots3, 0\ldots3, 0.75\frac{m}{sc^2}, 0.4sec)$

$v_{max}.T = 0.32m$. For this distance, the maximum error is $max(d_{error}) = 0.08m$. Thus, the $max(d_{error})$ in the reactive navigation context is low, and it can be neglected for collision checking. This error is comparable with other errors such as uncertainties in the sensor measurements, the real controllers, controller time delays, and communications time delays among others.

This statement is used to extend the EDT to the two-dimensional case with generality.

Appendix E

The Robots

E.0.4 Nomadic XR4000

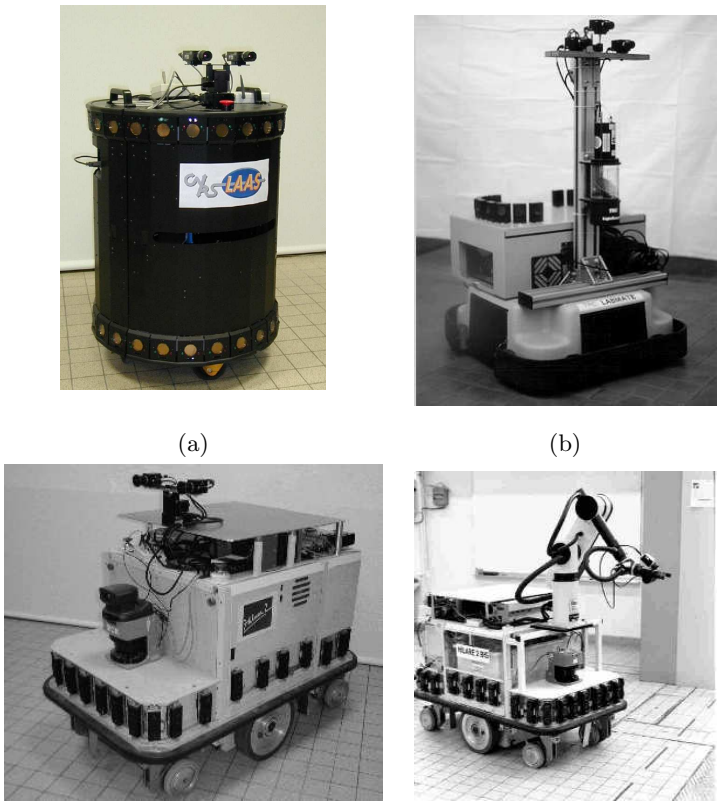
The *Nomadic XR400* is a circular and holonomic robot. The robot diameter is $0.7m$. The robot moves at omnidirectional translational velocities up to $1.2\frac{m}{sec}$, and accelerations up to $1.5\frac{m}{sec^2}$. It is equipped with a *SICK* 2-D laser rangefinder with a field of view of 180°, a range of 32 meters, an accuracy of up to $1cm$. Each laser scan has 361 points. See Fig. E.1a.

E.0.5 Labmate

The *Labmate* platform is an square and differential-driven robot. The robot side is $0.8m$. The maximum speed is $1\frac{m}{sec}$. It is equipped with a *TRC* 3D laser rangefinder that scans the environment with a maximum range of $6.5m$, and an accuracy up to $2.5cm$. See Fig. E.1b.

E.0.6 Hilare2 and Hilare2Bis

The *Hilare2* and *Hilare2Bis* platforms are rectangular and differential-driven. The robots are $1.3m \times 0.8m$. The maximum speed is $1.5\frac{m}{sec}$. Both robots are equipped with a SICK laser rangefinder, see Subsection E.0.4. The main difference between both robots is that *Hilare2Bis* is equipped with a GT6A arm with 6 degrees of freedom. The arm weighs about 60kg and it is located in the front part of the robot. This produces a high inertia when the robot turns. See Fig. E.1c and Fig. E.1d.



(a) (b)

(c) (d)

Figure E.1: a) Nomadic XR4000. b) Labmate platform. c) Hilare2. d) Hilare2bis.

E.0.7 Nomadic Scout

The *Nomadic Scout* is a circular and differential-driven robot. The robot diameter is $0.4m$. The robot moves up to $v_{max} = 1\frac{m}{sec}$. This base has a ring of 16 *Polaroid* ultrasounds, and it is equipped with a SICK laser (see Subsection E.0.4). See Fig. E.2a.

E.0.8 Lama

The *Lama* platform is has 6 driving wheels that can be programmed to work in differential-drive mode. The robot is rectangular ($1.85m \times 1.2m$). The maximum robot speed is $0.17\frac{m}{sec}$. It is equiped with a pair of B/W cameras. See Fig. E.2b.

(a) (b)

Figure E.2: a) Nomadic Scout. b) Lama.