Nearness Diagram (ND) Navigation: Collision Avoidance in Troublesome Scenarios

Javier Minguez, *Associate Member, IEEE,* and Luis Montano, *Member, IEEE*

***Abstract*—This paper addresses the reactive collision avoidance for vehicles that move in very dense, cluttered, and complex scenarios. First, we describe the design of a reactive navigation method that uses a "divide and conquer" strategy based on situations to simplify the difficulty of the navigation. Many techniques could be used to implement this design (since it is described at symbolic level), leading to new reactive methods that must be able to navigate in arduous environments (as the difficulty of the navigation is simplified). We also propose a geometry-based implementation of our design called the *nearness diagram navigation*. The advantage of this reactive method is to successfully move robots in troublesome scenarios, where other methods present a high degree of difficulty in navigating. We show experimental results on a real vehicle to validate this research, and a discussion about the advantages and limitations of this new approach.**

***Index Terms*—Collision avoidance, mobile robots, reactive navigation, sensor-based motion planning.**

I. Introduction

THERE ARE a lot of tasks where robots are asked to move safely in scenarios with unknown and dynamic obstacles. In this case, the motion strategies must rely on sensory information to compute the movements according to the unforeseen circumstances. These strategies are the sensor-based motion planning methods (also named reactive navigation methods). The challenge for these approaches is to deal with very cluttered, dense, and complex scenarios, which are usually the case in most robotic applications. A typical scenario is depicted in Fig. 1, where the robot is required to move among random distributions of obstacles with any shape, such as humans, doors, chairs, tables, wardrobes, and filing cabinets. Many existing reactive navigation methods have problems moving a robot in this type of environment. In this paper, we present how to use a classic paradigm to design a reactive navigation method, and we describe a particular implementation of this design that overcomes these navigation difficulties.

The *situated-activity* paradigm (see [1]) is a design methodology based on identifying situations and applying the corresponding actions. We use this methodology to design at symbolic level our reactive navigation method. Then, we simplify the navigation problem by a "divide and conquer" strategy based on a set of complete and exclusive situations. Therefore, reactive navigation methods implemented following our design guidelines must be able to solve more complex navigation problems than other existing methods (i.e., to successfully navigate in troublesome scenarios).

Fig. 1. Typical office environment. The snapshot was taken in an experiment performed using a complete navigation system. The ND navigation is the sensory-motor function that is driving the robot out of the office.

We call the *nearness diagram (ND) navigation* the geometric implementation of our design. By using some diagrams, entities as the proximity of obstacles and areas of free space are identified and used to define the set of situations, and to implement laws of motion (actions) for each situation. In real time, the sensory information is used to identify one situation, and the associated action is executed computing the motion commands. We validated our implementation with experimentation on a real vehicle in the mentioned scenarios.

Navigation in these enviroments (Fig. 1) remains troublesome for many existing methods, due to the appearance of classic problems such as trap situations in U-shape obstacles, oscillatory motion in narrow places, the difficulty of obtaining maneuvers that require motion toward the obstacles or far from the goal direction, or the identification of areas of motion where the robot could move without collisions. This paper describes how these difficulties are avoided by the ND navigation method. Moreover, we compare this method with other existing approaches on the basis of these limitations and problems.

In this paper, we discuss related work in Section II, and the situated-activity design methodology in Section III. We present the reactive navigation method design in Section IV, and our

Manuscript received September 27, 2002. This paper was recommended for publication by Associate Editor N. Sarkar and Editor A. De Luca upon evaluation of the reviewers' comments. This work was supported in part by the Ministerio de Ciencia y Tecnología del Gobierno España under MCYT-DPI2000-1272. This paper was presented in part at the IEEE/RSJ International Conference on Intelligent Robots and Systems, Takamatsu, Japan, October 31–November 5, 2000.

The authors are with the Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, 50015 Zaragoza, Spain (e-mail: jminguez@unizar.es; montano@unizar.es).

Digital Object Identifier 10.1109/TRA.2003.820849

implementation in Section V. In Section VI, we show the experimental results. Finally, in Section VII, we discuss the contributions and limitations of our reactive method, and in Section VIII, we draw our conclusions.

II. Related Work

The objective of our work is to compute collision-free motion for a robot operating in dynamic and unknown scenarios. Roughly, the motion techniques are either *global* and based on *a priori* information (motion planning), or *local* and based on *sensory information* (reactive navigation).

The theoretical aspect of the motion planning problem is well understood, and classically solved by computing a geometrical trajectory avoiding known obstacles (see [2] for a review of techniques). However, the general methods for motion planning are not applicable if the environment is dynamic with *a priori* unknown behavior, or if it is gradually discovered. Moreover, when both the environment model and robot motion are uncertain (as in the real world, because of sensing inaccuracies), executing a theoretical geometric trajectory is not realistic and the robot is doomed to collide with obstacles.

Hence, solving this problem involves sensing directly within the motion planning and control loop. Reactive navigation is, then, a more robust way to tackle the mobility problem by taking into account the reality of the environment during motion. These methods are based on a *perception-action* process that is repeated periodically at a high rate. First, the sensory information is collected. Then, these methods compute the "best" motion command to avoid collisions while moving the robot toward a given goal location. This process is resumed while the vehicle executes the motion command. These methods potentially deal with unknown and dynamic scenarios because the sensory information is integrated at a high rate within the framework. However, it is difficult to obtain optimal solutions and to avoid the trap situations since they use a local fraction of the information available (sensory information). Next, we describe related work with these methods.

- Some methods use a physical analogy to compute the motion commands, where mathematical equations borrowed from physics are applied to the sensory information and the solutions are transformed into motion commands (e.g., the potential field methods [3]–[8], the perfume analogy [9], and the fluid analogy [10], among others).
- Some methods compute a set of suitable motion commands to select one command based on navigation strategies. Some methods calculate sets of steering angles (e.g., [11]–[14]), and others compute sets of velocity commands (e.g., [15]–[18]).
- Other methods compute some form of high-level information description from the sensory information to obtain a motion command later on (e.g., [6], [19], [20]). The ND navigation method belongs to this group of approaches, since some intermediate entities are computed to select a given situation, and then an action that computes the motion is executed.

The majority of these methods have a high degree of difficulty in safely navigating in very dense, cluttered, and complex scenarios. Navigation in these circumstances is the motivation and objective of our work.

III. The Situated-Activity Paradigm of Design

The *situated-activity* paradigm of behavioral design [1] was used to design our reactive navigation method. This paradigm is based on defining a set of situations that describe the relative state of the problem entities, and on actions associated with each situation. During the execution phase, perception is used to identify the current situation and the associated action is carried out.

A design based on this paradigm has to comply with some **requirements**.

- The situations have to be *identifiable* from sensory perception, *exclusive*, and *complete* to represent the relative state of the problem entities. Moreover, an explosion in the number of situations needed has to be avoided. As commonly pointed out (see [21]), the most difficult step is to find a set of situations that effectively describes the task.
- Each action design has to solve the task problem *individually* in the context of each situation.

Using this paradigm to design a module that executes action tasks based on sensory information has the following **advantages**.

- The paradigm itself describes perception-action process.
- The paradigm itself is a "divide and conquer" strategy based on situations to reduce the task difficulty.
- A design using this paradigm does not have the real-time *action coordination problem*[1] because it is based on a *complete* and *exclusive* set of situations (so there is not ambiguity in the action selection).

IV. The Reactive Navigation Method Design

We describe in this section how we used the *situated-activity* paradigm to design a reactive navigation method that works as follows (Fig. 2). Periodically the sensory information collected is used to identify the current situation among the predefined set (Section IV-A), and then, the associated action is executed computing the motion (Section IV-B).

A. The Set of Situations

The objective is to describe the relative state of the reactive navigation entities (i.e., the robot, the obstacle distribution, and the goal location) with a set of situations. First, we analyze the relations among the entities to define the situations.

We obtain the relation between the robot and the obstacle distribution with a safety evaluation [we check whether there are obstacles within a security zone around the robot bounds, see Fig. 3(a)]. In addition, we use an intermediate device, the *free walking area*, to relate the robot and goal locations by means of the obstacle distribution structure. The free walking area is computed as follows. First, we search for gaps in the obstacle distribution, and obtain the *regions* from two contiguous gaps.

[1]In short, this problem arises when the main task is divided into subtasks that have to be arbitrated by a decision algorithm (to decide which subtask is active in real time).

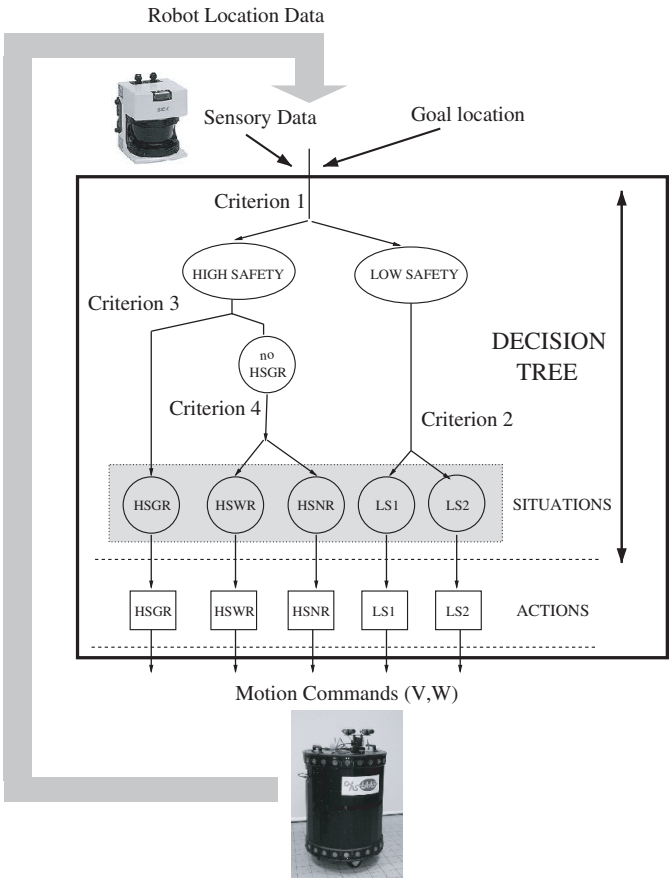

Fig. 2. Reactive navigation method design.

Next, we select the closest region to the goal location, checking whether is "navigable" by the robot [Fig. 3(a)].

Next, we use these relations to define the set of situations that are represented in a *decision tree* (Fig. 2). The inputs of the tree are the robot, the obstacle distribution (sensory information), and the goal, which allow us to identify the current situation (output). The tree is traversed using binary decision rules based on criteria that depend on the inputs and their relations. We describe the four criteria below.

Criterion 1: Safety criterion. There are two safety situations (Fig. 2), depending on whether there are obstacles within the *security zone* [Low Safety, see Fig. 3(a) and (b)] or not [High Safety, see Fig. 3(c), (d), and (e)]. In Low Safety, we obtain the first two situations by applying the next criterion.

Criterion 2: Dangerous obstacle distribution criterion.

1) **Low Safety 1 (LS1)**: The robot is in LS1 when the obstacles in the security zone are only on one side of the gap (closest to the goal) of the free walking area [Fig. 3(a)].
2) **Low Safety 2 (LS2)**: The robot is in LS2 when the obstacles in the security zone are on both sides of the gap (closest to the goal) of the free walking area [Fig. 3(b)].

There are three situations in High Safety. We obtain the first one by applying the following criterion.

Criterion 3: Goal within the free walking area criterion.

3) **High Safety Goal in Region (HSGR)**: The robot is in HSGR when the goal location is within the free walking area [Fig. 3(c)].

If not, we obtain the last situations by applying the next criterion.

Criterion 4: Free walking area width criterion. A free walking area is wide if its angular width is larger than a given angle, and narrow, otherwise.

4) **High Safety Wide Region (HSWR)**: The robot is in HSWR when the free walking area is wide [Fig. 3(d)].
5) **High Safety Narrow Region (HSNR)**: The robot is in HSNR when the free walking area is narrow [Fig. 3(e)].

These situations are identifiable from sensory perception, when it is available as depth maps. They are exclusive and complete because they are represented with a binary decision tree. In addition, there is no explosion in the number of situations because there are only five. This is because the situation definition does not depend on the resolution or size of the space considered. Then, we conclude that the set of situations comply with the requirements imposed by the situated-activity paradigm (mentioned in Section III).

B. Action Design

We describe next the action design guidelines associated with each situation.

1) **Low Safety 1 (LS1)**: This action moves the robot away from the closest obstacle, and toward the gap (closest to the goal) of the free walking area [Fig. 3(a)].
2) **Low Safety 2 (LS2)**: Centers the robot between the two closest obstacles at both sides of the gap (closest to the goal) of the free walking area, while moving the robot toward this gap [Fig. 3(b)].
3) **High Safety Goal in Region (HSGR)**: Drives the robot toward the goal [Fig. 3(c)].
4) **High Safety Wide Region (HSWR)**: Moves the robot alongside the obstacle [Fig. 3(d)].
5) **High Safety Narrow Region (HSNR)**: Directs the robot through the central zone of the free walking area [Fig. 3(e)].

In each situation, the action individually solves the reactive navigation task, which is to avoid obstacles while moving the robot toward the goal location. This is achieved in Low Safety because both actions avoid the obstacles while moving the robot toward the gap (closest to the goal) of the free walking area (notice that this gap implicitly has information about the goal location). In High Safety, there is no need to avoid collisions because the robot is not in danger. The actions drive the robot toward the goal, toward the gap (closest to the goal) of the free walking area, or toward the central zone of the free walking area (i.e., these actions explicitly or implicitly drive the robot toward the goal location). Then, the design of the actions comply with the requirements imposed by the situated-activity paradigm (mentioned in Section III).

There are some points worth mentioning here.

1) The reactive navigation method design is described at the symbolic level. Learning techniques, fuzzy sets, potential field implementations, optimization techniques, and other tools may be used to implement the design, leading to new reactive navigation methods. We describe in the next section a geometry-based implementation.

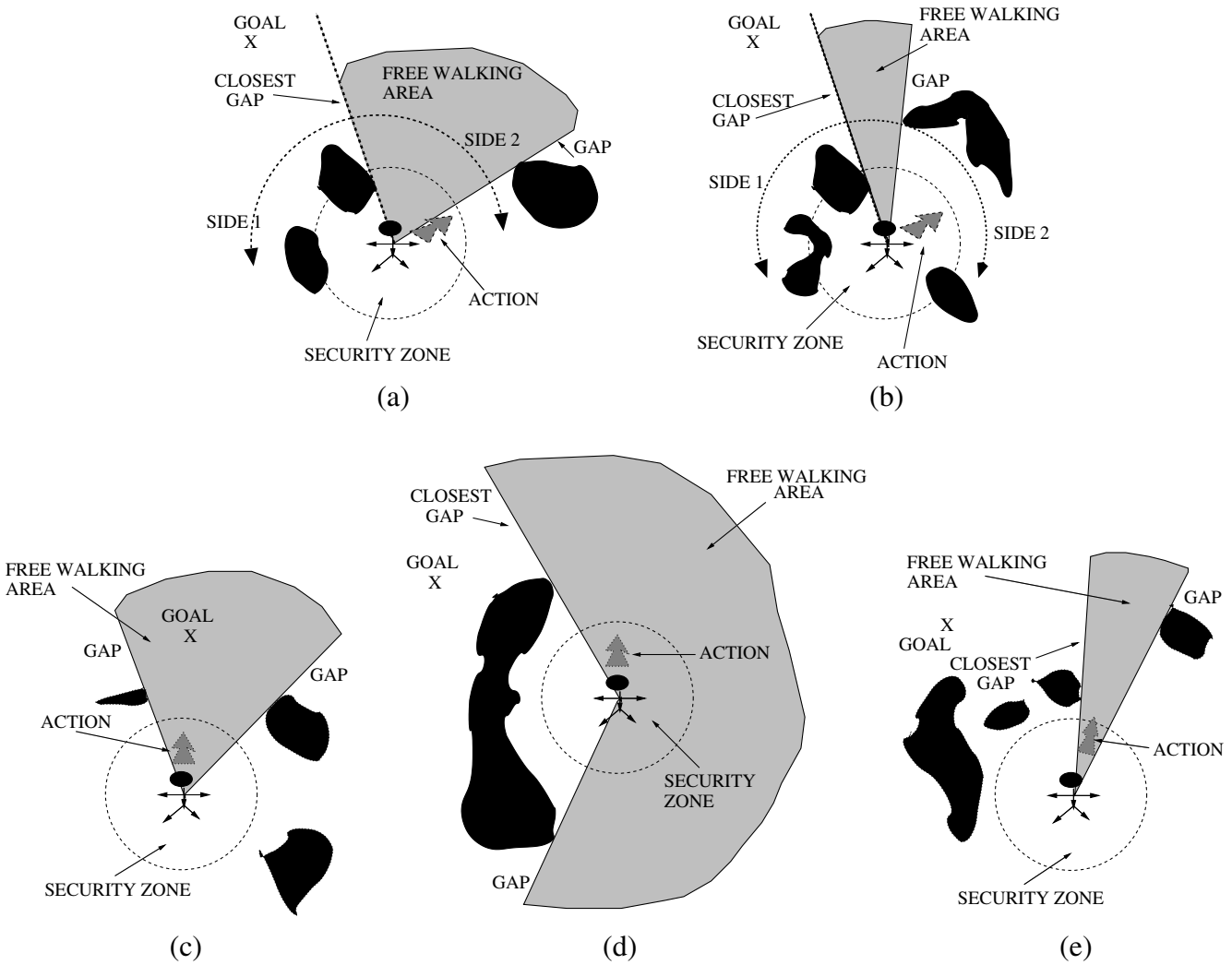

Fig. 3. (a) LS1 situation/action example. (b) LS2 situation/action example. (c) HSGR situation/action example. (d) HSWR situation/action example. (e) HSNR situation/action example.

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 complicated navigation problems than other methods (since the majority of them usually use a unique navigation heuristic). In addition, the design is flexible and new situations could be defined to simplify even further.
3) The design does not suffer from the "action coordination problem." The actions are self-coordinated because the general situations are complete and exclusive. Then, only one situation is selected each time and only one action is executed.

In summary, we have presented in this section the design of a reactive navigation method using the situated-activity paradigm, and demonstrated that the design complies with the requirements imposed by the paradigm. Next, we implement the design.

V. ND Navigation

We describe here a geometry-based implementation of the reactive navigation method design called ND navigation. We consider a circular (with radius R) and holonomic vehicle that moves over a flat surface. The workspace $\mathcal{W}$ is $\mathbb{R}^2$, and a motion command is $(\mathbf{v}, w)$ (with $\mathbf{v} = (v_m, \theta)$ the translational velocity, and w the rotational velocity).

We assume that the sensory information is available as depth point maps to maintain the sensor as generally as possible (the great majority of sensory information can be processed and then reduced to points), and to avoid the use of structured information (as lines or polygons that otherwise can be used if they are available).

In order to implement the ND method (Fig. 2), first we introduce the tools used to analyze the information (Section V-A). Next, we present the implementation of the set of situations (Section V-B), and of the associated actions (Section V-C).

A. Tools and the Relations Among the Navigation Entities

The NDs are the tools used to analyze the relations between the robot, obstacle distribution, and goal location.

From now on, the reference system is the robot reference. We divide the space in n sectors centered in the origin (in our

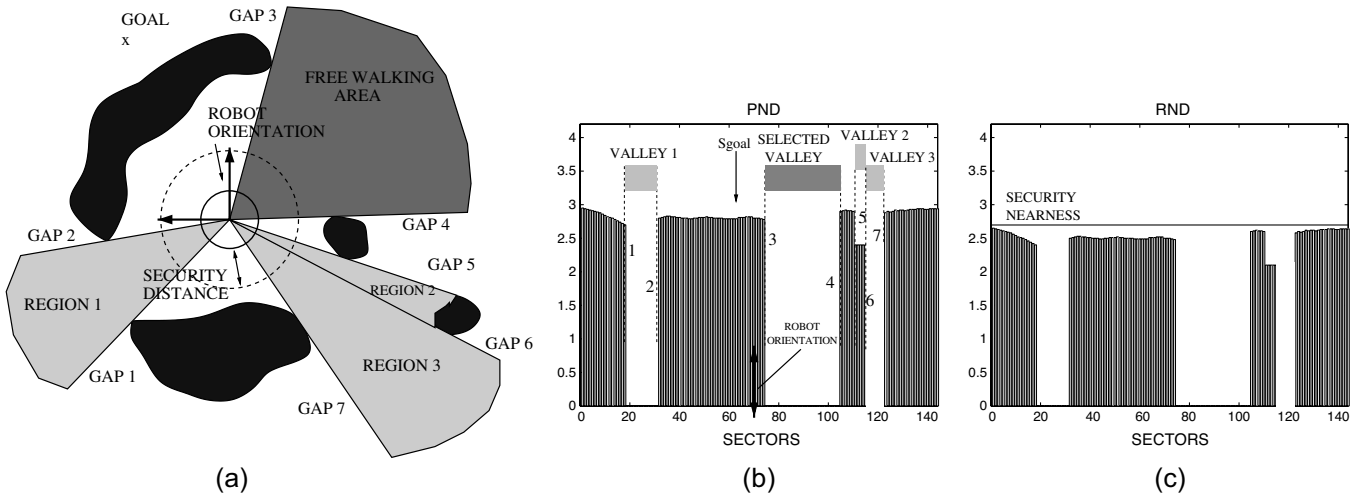

Fig. 4. (a) Gaps, regions, and free walking area. (b) PND. (c) RND. The following values were set: $R = 0.3$ m, $d_{\max} = 3$ m, $d_s = 0.3$ m.

implementation $n = 144$, so 2.5° is the angle of each sector). Once a sector s is selected, we compute the bisector angle as

$$\theta = \text{bisec}(s) = \pi - \frac{2.\pi}{n} \cdot s. \tag{1}$$

Let L be the list of the obstacle points perceived, then $\delta_i(L)$ is the function that computes the minimum distance to an obstacle point in sector i (with $\delta_i(L) = 0$ when there are no obstacles in sector i, and $\max(\delta_i(L)) = d_{\max}$ where $d_{\max}$ is the maximum range of the sensor). Then we define the diagrams as follows.

Definition 1: ND from the central Point (PND)

$$\begin{aligned} \text{PND}: \quad & \{0,\ldots,n-1\} \rightarrow \mathbb{R}_0^+ \\ & i \rightarrow \text{PND}_i \\ \text{if } \delta_i(L) > 0, \quad & \text{PND}_i = d_{\max} + 2R - \delta_i(L) \\ \text{else} \quad & \text{PND}_i = 0. \end{aligned}$$

Definition 2: ND from the Robot bounds (RND)

$$\begin{aligned} \text{RND}: \quad & \{0,\ldots,n-1\} \rightarrow \mathbb{R}_0^+ \\ & i \rightarrow \text{RND}_i \\ \text{if } \delta_i(L) > 0, \quad & \text{RND}_i = d_{\max} + E_i - \delta_i(L) \\ \text{else} \quad & \text{RND}_i = 0. \end{aligned}$$

where E_i is the robot radius (R) for a circular robot.[2]

The PND represents the nearness of the obstacles from the robot center and the RND represents the nearness of the obstacles from the robot boundary (see Fig. 4). Next, we consider the relations among the robot, the obstacle distribution, and the goal using these diagrams.

We obtain the robot and obstacle distribution relation by checking whether there are obstacles within the security zone (defined with a *security distance*, d_s, to the robot bounds). Then, we use a *security nearness* (computed by $n_s = d_{\max} - d_s$) in the RND to evaluate the robot safety [Fig. 4(a) and (c)].

The relation between the robot and the goal location is obtained from the free walking area device. We carry out the following analysis in the PND to identify it. First we identify gaps, and from these gaps, we obtain the regions. Finally, we select one region, the free walking area [Fig. 4(a)].

1) *Gaps*: We identify gaps in the obstacle distribution as *discontinuities* in the PND.

   A discontinuity exists between two adjacent[3] sectors (i, j) if $|\text{PND}_i - \text{PND}_j| > 2R$. Fig. 4(a) depicts the gaps that are identified as discontinuities in the PND [Fig. 4(b)]. Notice that the robot diameter $(2R)$ is used because we are only interested in the gaps where the robot fits.

   For a discontinuity between two sectors (i, j), if, for instance, $\text{PND}_i > \text{PND}_j$, we differ between a *rising discontinuity* from j to i and a *descending discontinuity* from i to j.

2) *Regions*: Two contiguous gaps form a region. We identify the regions as *valleys* in the PND.

   Let $S = \{0,\ldots,n-1\}$ be the set of all sectors. A valley is a nonempty set of sectors of S, $V = \{l,\ldots,r\}$, that satisfies the following conditions.

   (a) There are no discontinuities between adjacent sectors of V (i.e., there are no discontinuities within the valley).

   (b) There are two discontinuities in the extreme sectors of V (l and r)

$$|\text{PND}_{l-1} - \text{PND}_l| > 2R \quad \text{AND} \quad |\text{PND}_{r+1} - \text{PND}_r| > 2R.$$

   (c) At least one of the previous discontinuities is a rising discontinuity from l or from r

$$\text{PND}_{l-1} > \text{PND}_l \quad \text{OR} \quad \text{PND}_{r+1} > \text{PND}_r$$

   where the rising discontinuities identify potential gaps to drive the robot within the region, so at least one is required.

   Fig. 4(a) shows the four regions identified as valleys in the PND [Fig. 4(b)]. These valleys do not have inside discontinuities [condition (a)], and they have a discontinuity in both extremes [condition (b)]. Furthermore, each valley has at least one rising discontinuity [condition (c)].

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

[3]The adjacent sectors to i are $i-1$ and $i+1$. In all the operations among sectors, we use the $\text{mod}(*, n)$ function to give continuity to the diagrams. Thus, if $i = n$, then $i = 0$.

For instance, valley 1 is created by discontinuities 1 and 2, both rising discontinuities (and identifies region 1 created by gaps 1 and 2). However, valley 2 is created by the rising discontinuity 5 and the descending discontinuity 6 (identifying region 2 created by gaps 5 and 6). Notice that the descending discontinuity 6 identifies the gap 6 that cannot be reached moving within the region 2. However, on the other side, discontinuity 6 is a rising discontinuity, and with discontinuity 7 creates the valley 3 (identifying region 3). A special case is when the goal is between an obstacle and the robot, then it could be that the sector that contains the goal location (s_{goal}) does not belong to a valley. When this situation is detected, we set $\mathrm{PND}_{s_{\mathrm{goal}}} = 0$, which creates an artificial valley in the goal sector (in this case, we force the goal to be within a region). Another special case is when there are no obstacles, and then all the sectors form the valley.

3) *Free walking area*: The "navigable" region closest to the goal location, which is identified as follows. We select first the valley with the rising discontinuity closest[4] to s_{goal} (in Fig. 4, we select the valley created by discontinuities 3 and 4, because discontinuity 3 is the rising discontinuity closest to s_{goal}). Next, we check whether the candidate region is "navigable" (see the Appendix for description of the algorithm). The selected valley identifies the free walking area. If it is not "navigable," we select another valley and repeat the process until we find a "navigable" region, or no region exists.

   We still need to differ between a wide free walking area and a narrow one. If its angular width is greater than a given quantity (for us, 90°) is wide, if not, it is narrow. Then, since the number of sectors of a valley is the angular width of the region, a valley is wide if the number of sectors is greater than $s_{\max} = (n/4)$ (that is, 90°), and narrow otherwise.

We adopt the following notation to simplify the description of the set of situations and the associated actions in the next subsections (see Fig. 5).

- s_{goal}: sector that contains the goal location.
- s_{rd} and s_{od}: s_{rd} is the sector corresponding to the *rising discontinuity* (closest to s_{goal}) of the selected valley. This sector contains the potential gap (closest to the goal) of the free walking area. s_{od} is the sector corresponding to the other discontinuity. This sector contains the other gap of the free walking area.
- s_{ml} and s_{mr}: RND sectors that exceed the *security nearness* at both sides of s_{rd} (with maximum values). These sectors contain the closest obstacle points at both sides of the potential gap (closest to the goal) of the free walking area.

B. Set of Situations

Using these tools, we address the implementation of the set of situations mentioned in Section IV-A. The situations are represented in the same *decision tree* of Fig. 2, and the criteria of the tree branches are described below.

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

Criterion 1: Safety criterion. To compute this criterion, we check whether there are obstacles that exceed the security nearness in the RND (Low Safety), or not (High Safety) (Fig. 5). In Low Safety, we obtain the first two situations by applying the next criterion.

Criterion 2: Dangerous obstacle distribution criterion.

2) **Low Safety 1 (LS1)**: The robot is in LS1 when the RND sectors that exceed the security nearness are only on one side of the rising discontinuity (closest to the goal sector) of the selected valley. We depict this situation in Fig. 5, where in LS1 there are RND sectors that exceed the security nearness, but only on one side of s_{rd}.
3) **Low Safety 2 (LS2)**: The robot is in LS2 when the RND sectors that exceed the security nearness are on both sides of the rising discontinuity (closest to the goal sector) of the selected valley. This case is similar to the previous one, but the RND sectors that exceed the security nearness are now at both sides of s_{rd} (see Fig. 5).

There are three situations in High Safety. We obtain the first one by applying the following criterion.

Criterion 3: Goal within the free walking area criterion.

3) **High Safety Goal in Region (HSGR)**: The robot is in HSGR if the goal sector (s_{goal}) belongs to the selected valley (see Fig. 5 and notice that the robot is in High Safety because no RND sector exceeds the security nearness).

If not, we obtain the last situations by applying the next criterion.

Criterion 4: Free walking area width criterion.

4) **High Safety Wide Region (HSWR)**: The robot is in HSWR when the selected valley is wide. We show this situation in Fig. 5, where the selected valley is wide (the number of sectors is $102 > (n/4) = 36$).
5) **High Safety Narrow Region (HSNR)**: The robot is in HSNR when the selected valley is narrow. In Fig. 5, the selected valley is narrow because the number of sectors is $13 < (n/4) = 36$.

C. Associated Actions

We describe next the implementation of the actions associated with each situation. The objective is to find simple control laws that produce the desired navigation behavior in each situation (following the design guidelines mentioned in Section IV-B). Each action computes a motion command (v_m, θ, w), whose implementation is summarized in Table I.

1) Translational Velocity Direction (θ): To compute the most promising motion direction, we carry out all the operations with sectors that are converted into an angle θ by using (1) (as $s_\theta \in \mathbb{R}$, then $\theta \in [-\pi, \pi]$, i.e., any direction of motion can be selected).

In Low Safety, the direction of motion must bring the robot to a secure situation, since the robot is in danger of colliding because there are obstacles within the security zone.

1) **Low Safety 1**: In this situation, the direction of motion is computed by adding two terms. The first one is s_{rd} as it implicitly contains the information of the goal location. The second one is a magnitude that depends on the angle between s_{rd} and the closest obstacle direction (s_{ml}), plus a

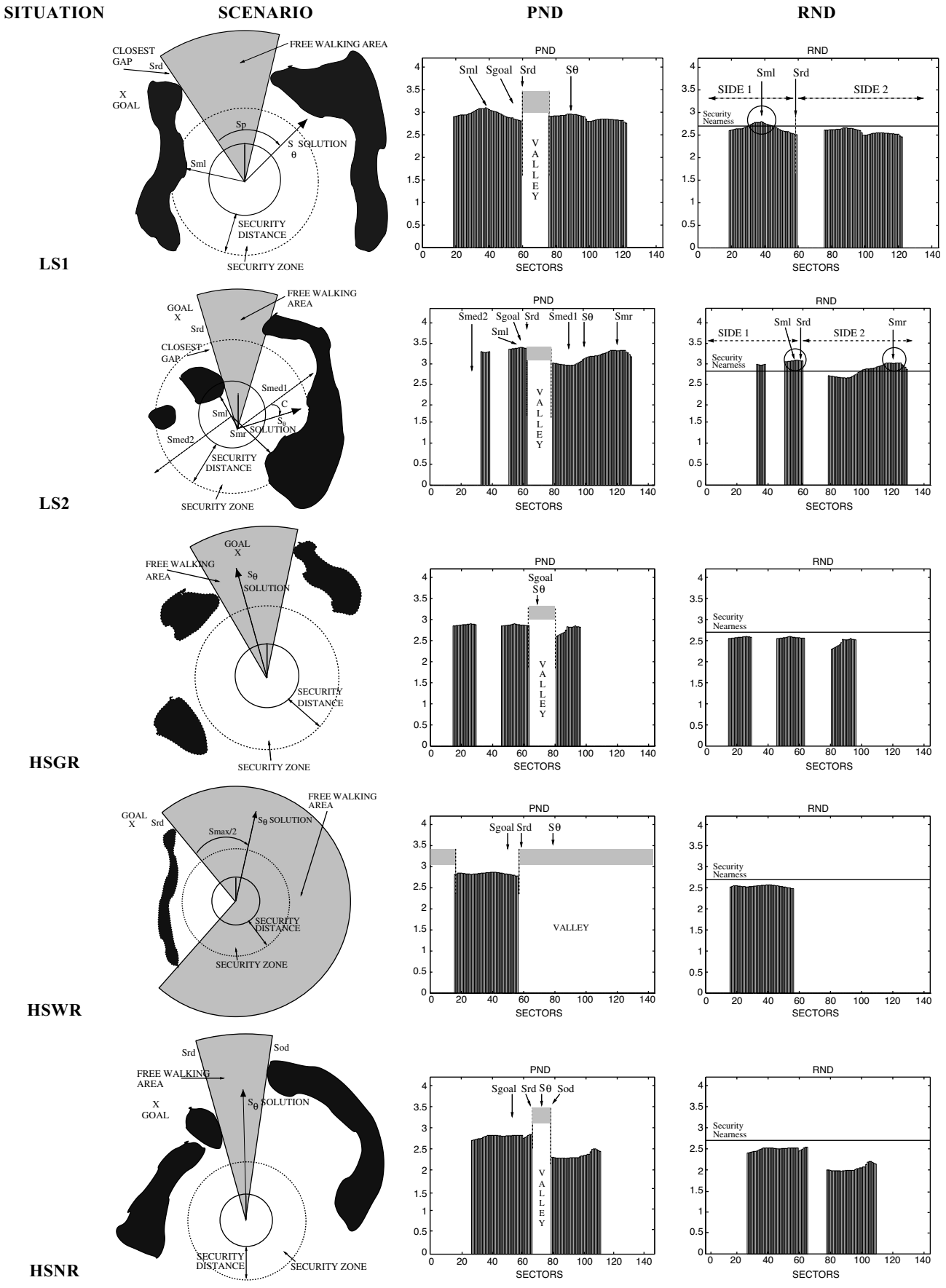

Fig. 5. Situation/Action table and the PND and RND diagrams.

TABLE I
SITUATION/ACTION TABLE

TABLE: Situation/Action	*ACTIONS*		
SITUATIONS	θ Trans. vel. angle ($\theta = bisec(s_\theta)$)	v_m Trans. vel. module	w Rot. vel.
Low Safety 1	$s_p = \lvert s_{rd} - s_{ml}\rvert * p + \frac{s_{max}}{2}$ $s_\theta = s_{rd} + sign(s_{rd} - s_{ml}) * s_p$	$v_m = v_{max} * \frac{d_{obs}}{d_s} * (\frac{\frac{\pi}{2} - \lvert\theta\rvert}{\frac{\pi}{2}})$	$w = w_{max} * \frac{\theta}{\frac{\pi}{2}}$
Low Safety 2	$s_{med_1} = \frac{s_{ml}+s_{mr}}{2}$ $s_{med_2} = \frac{s_{ml}+s_{mr}+n}{2}$ If $\lvert s_{rd} - s_{med_1}\rvert < \lvert s_{rd} - s_{med_2}\rvert$ then $s_\theta = s_{med_1} \pm c$ Else $s_\theta = s_{med_2} \pm c$		
High Safety Goal in Region	$s_\theta = s_{goal}$	$v_m = v_{max} * (\frac{\frac{\pi}{2} - \lvert\theta\rvert}{\frac{\pi}{2}})$	
High Safety Wide Region	$s_\theta = s_{rd} \pm \frac{s_{max}}{2}$		
High Safety Narrow Region	$s_\theta = \frac{s_{rd}+s_{od}}{2}$		

fixed angle ($s_{\max}/2$). The addition of both terms leads to a motion behavior toward $s_{\rm rd}$ while avoiding the closest obstacle (see Fig. 5).

The p parameter is an experimentally tuned parameter (in our implementation $p \in [1.5, 2.5]$) whose value ensures a smooth behavior in the transitions among the situations. The parameter acts as an adaptive proportional controller.

2) **Low Safety 2**: The direction solution is computed as the bisector of the direction of the two closest obstacles ($s_{\rm ml}$ and $s_{\rm mr}$). From the bisector and the complementary angle, we choose the closest to ($s_{\rm rd}$), as it implicitly contains the information of the goal location. In addition, we add a term c that is a correction function used to keep the robot centered between the two closest obstacles, since motion along the bisector does not center the robot. 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 depending on the sector ($s_{\rm ml}$ or $s_{\rm mr}$) that contains the closest obstacle.

In High Safety, we move the robot within the free walking area because the robot is not in danger of colliding.

3) **High Safety Goal in Region**: In this situation, the direction of motion is toward the goal location. We explicitly use the goal to compute the motion commands only in this situation (notice that in this situation the robot is not in danger of colliding and the goal is within the free walking area).
4) **High Safety Wide Region**: The direction of motion is the addition of $s_{\rm rd}$ (that contains information of the goal) and a given angle ($s_{\max}/2$). This produces a motion alongside the obstacle toward the goal.
5) **High Safety Narrow Region**: The direction is computed as the bisector of the direction of the discontinuities of the selected valley, that is, toward the central zone of the free walking area.

2) Translational Velocity Absolute Value (v_m): Let $v_{\max}$ be the maximum translational velocity, $d_{\rm obs}$ be the distance from the closest obstacle to the robot bounds, and d_s be the security distance. Then, with the proposed velocity control (Table I), the robot moves at maximum speed (High Safety) until one obstacle shows up in the security zone. Then, the robot reduces the speed in proportion to the distance to the closest obstacle (Low Safety), until the security zone is clear. In addition, large changes in the direction of motion also reduce the translational velocity module. Notice that since we will use a sensor with 180° visibility, we prohibit instantaneous backward motion and force the velocity direction to be $\theta \in [-\pi/2, \pi/2]$. Then, large changes in the direction of motion also reduce the translational velocity module.

3) Rotational Velocity (w): We introduce this angular velocity term because the sensor has visibility constraints (it is considered that the main sensor direction and the robot orientation match). This angular velocity control (Table I) aligns the main sensor direction with the robot instantaneous direction of motion, with large 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.

In summary, in this section we have presented the ND method that is a geometry-based implementation of the reactive method design. This reactive method computes the motion commands (v_m, θ, w) from the sensory information, in order to safely drive a vehicle among locations.

VI. SETTINGS AND EXPERIMENTAL RESULTS

In this section, we present the experimental results to validate the ND method.

A. Mobile Platform and Settings

We tested the ND method on a Nomadic XR4000, a circular (with a radius of 24 cm) and holonomic vehicle equipped with a 2-D laser rangefinder and an on-board Pentium II. The computation time of the ND method was around 125 ms when processing a short-time memory (required to deal with the sensor visibility constraints) built with the last 20 laser measurements (361 × 20 points). We set the maximum translational velocity to $v_{\max} = 0.3$ m/s and the maximum rotational one to $w_{\max} = 1.57$ rad/s. We fixed these velocity limits because the intention was to move the robot in indoor human environments, where there is a high density of obstacles and the safety of the humans around must be preserved (see Fig. 1).

B. Experiments

We present here three experiments carried out using this vehicle in unknown, unstructured, and dynamic scenarios (only the goal location was given in advance). In addition, we designed the experiments to verify that the ND method complies with the goal of this work: *To safely drive a robot in very dense,*

cluttered, and complex scenarios. Furthermore, these experiments will also allow a discussion (next section) of some other contributions of the ND method summarized next: *i*) avoiding trap situations due to the perceived environment structure (e.g., U-shaped obstacles and two very close obstacles); *ii*) computing stable and oscillation-free motion; *iii*) selecting motion directions toward obstacles; *iv*) exhibiting a high goal insensitivity (i.e., to be able to choose motion directions far away from the goal direction); and *v*) selecting regions of motion using a robust "navigable" criterion.

Experiment 1: In this experiment, the robot reached the goal location in a dense scenario, with narrow places and highly reduced room to maneuver [see the robot trajectory and the laser points perceived in Fig. 6(a) and the sequence of snapshots Fig. 6(c)–(j)]. Fig. 6(d), (f), and (g) depict some parts of the experiment where the robot moved among obstacles with less than 10 cm on both sides (the tile size is about 10 cm). In addition, we did not observe trap situations due to the motion in narrow places.

The robot was able to enter and travel along the passage because the available space was checked with the free walking areas [some of them are shown in Fig. 6(e), (k), (f), and (l)]. The selection of directions toward the obstacles was essential to successfully accomplish this experiment (in Fig. 6(e), (k), (f), (l), (g), and (m) we depict some instants when the computed direction solution pointed toward an obstacle). The motion was oscillation free, which is illustrated in the robot path and in the velocity profiles [see Fig. 6(a) and (b)]. We show in Fig. 6(n) the situation selected at each time, where the robot was mainly in LS2 because there were obstacles within the security zone on both sides of the free walking area at every moment.

The experiment was carried out in 60 s, and the average translational velocity was 0.114 m/s.

Experiment 2: The robot navigated in a dense, complex, and cluttered scenario that was dynamically built by a human while the robot was moving [the sequence of snapshots show the highly dynamic nature of the environment in Fig. 7(c)–(j)]. In the first part of the experiment, the human closed the passage when the robot was in the first corridor [Fig. 7(f)]. This situation was detected and the robot was stopped (notice that in Fig. 7(b) the velocities from second 33 to 52 are zero). Here, a flag could be launched to a higher level module to plan a new subgoal, however, in these experiments, we only tested the reactive method. Finally, the passage was opened and the robot resumed the motion [Fig. 7(g)].

In some parts of the experiment, the robot navigated among very close obstacles [see Fig. 7(e), (k), (g), and (l)], where we did not detect trap situations. The selection of areas of motion where the robot fitted was carried out using the free walking area [see some of them in Fig. 7(e), (k), (h), and (m)]. The ND method selected motion directions toward the obstacles when it was required [some of them are illustrated in Fig. 7(e), (k), (g), (l), (h), and (m)]. To successfully navigate in this environment, the method selected directions of motion far away from the goal direction (mentioned before as goal insensitivity). Fig. 7(g), (l), (h), and (m) depict some instants when the motion direction solutions and goal directions differ in more than 90° (any difference could be obtained with the reactive method). We did not observe oscillations during the run, which is illustrated by the robot path and the velocity profile [Fig. 7(a) and (b)].

We show the situations selected in Fig. 7(n), where the robot mainly was in LS2 because there were obstacles within the security zone on both sides of the free walking area at almost every moment. Sometimes the robot was in LS1 because there were risky obstacles only on one side of the free walking area. This is due to 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 became LS2).

The complete time of the experiment was 220 s and the average translational velocity was 0.104 m/s.

Experiment 3: In this experiment, the robot reached the goal location avoiding three U-shape obstacles placed in the environment [see Fig. 8(a) and Fig. 8(c) and (g)]. The robot avoided entering and getting trapped because the areas of motion selected (free walking areas) were out of the U-shaped obstacles (i.e., the ND method uses the free walking area device to avoid these structural trap situations). Fig. 8(c), (k), (e), (l), (g), and (m) depict some parts of the experiment and the free walking areas that, in all the cases, are out of the U-shaped obstacles. The free walking area of Fig. 8(m) has the same shape as region 1 in Fig. 4, while the ones of Fig. 8(k) and (l) have the same shape as region 2.

Directions toward the obstacles were selected during almost the whole experiment (see in Fig. 8(e), (l), (g), and (m) some of these moments). In some parts of the experiment, motion directions far from the goal direction were required [Fig. 8(m)].

The velocity profile is illustrated in Fig. 8(b). These velocities are higher than in other experiments because the robot was often in High Safety [Fig. 8(n)] and then moved at maximum speed. The time of the experiment was 83 s and the average translational velocity was 0.247 m/s.

VII. Discussion

We present here a discussion regarding other collision avoidance approaches, the limitations of the ND method and the limitations of the reactive approaches in general.

A. Comparison With Existing Methods

The ND method avoids the **local trap situations** due to the environmental structure (e.g., U-shaped obstacles and very close obstacles). The method successfully selects areas of motion among very close obstacles because the free walking areas are selected with a width-checking criterion [see Fig. 6(d), (k), (f), and (l) and Fig. 7(e), (k), (h), and (m)]. In addition, there are no free walking areas within a U-shaped obstacle when it is completely "visible." In this case, the free space within the obstacle is not selected for motion [see Fig. 8(c), (k), (e), (l), (g), and (m)]. Sometimes the free walking area could be within an obstacle when it is not completely "visible." In this case, some symmetrical conditions involving the goal location would produce motion toward the inside of the obstacle.

The potential field methods produce local trap situations due to the motion among close obstacles and the U-shaped obstacles [22] (both cases create potential minima that trap the robot).

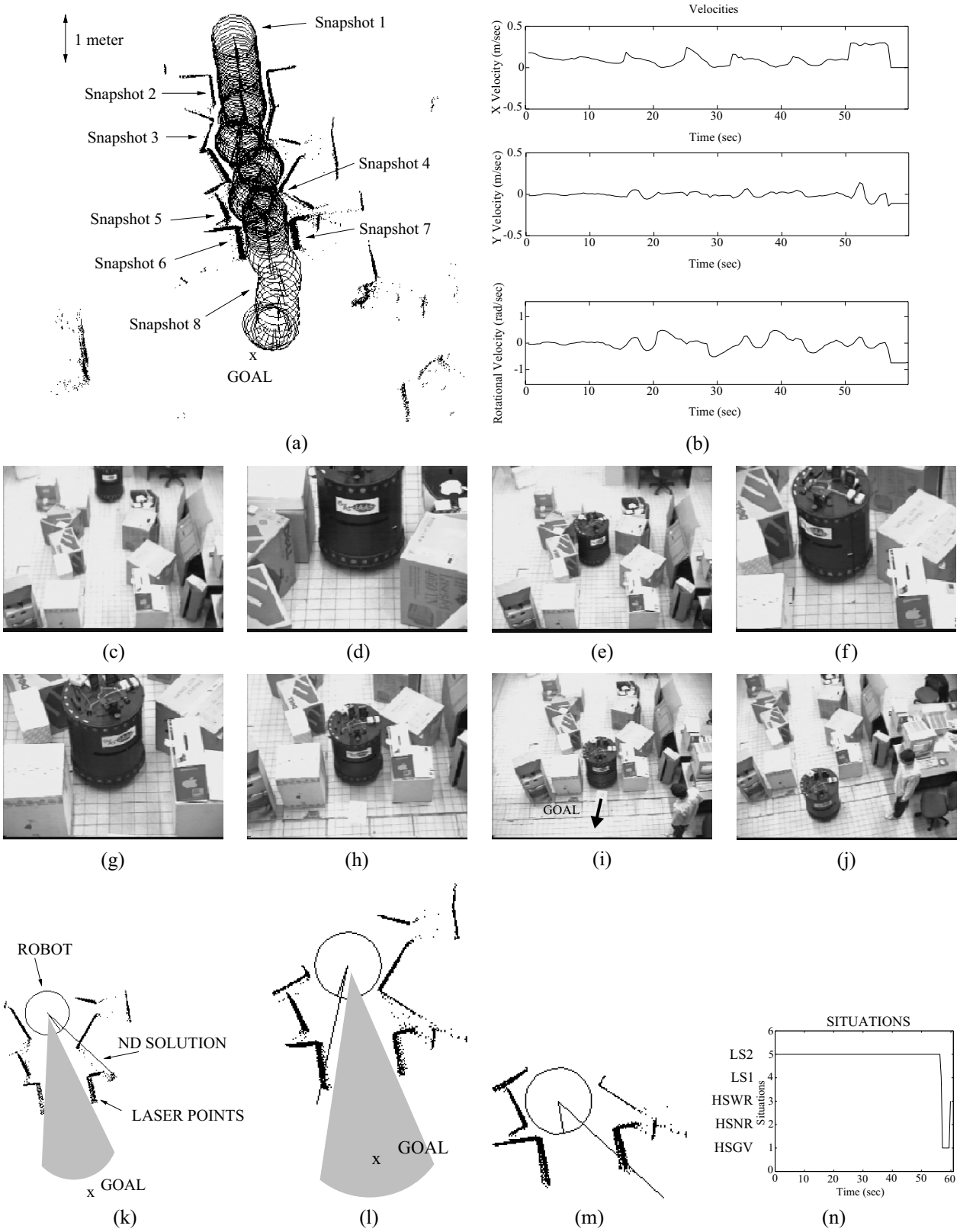

Fig. 6. Experiment 1.

To move a robot among close obstacles, the methods based on polar histograms [12], [13], [23] 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-shaped obstacles are not avoided by the methods that use constrained optimizations [11], [15], [16], [18]. This is because the optimization loses the information of the environment structure that is necessary to solve these situations (the environment structure is studied with the free walking area in the ND

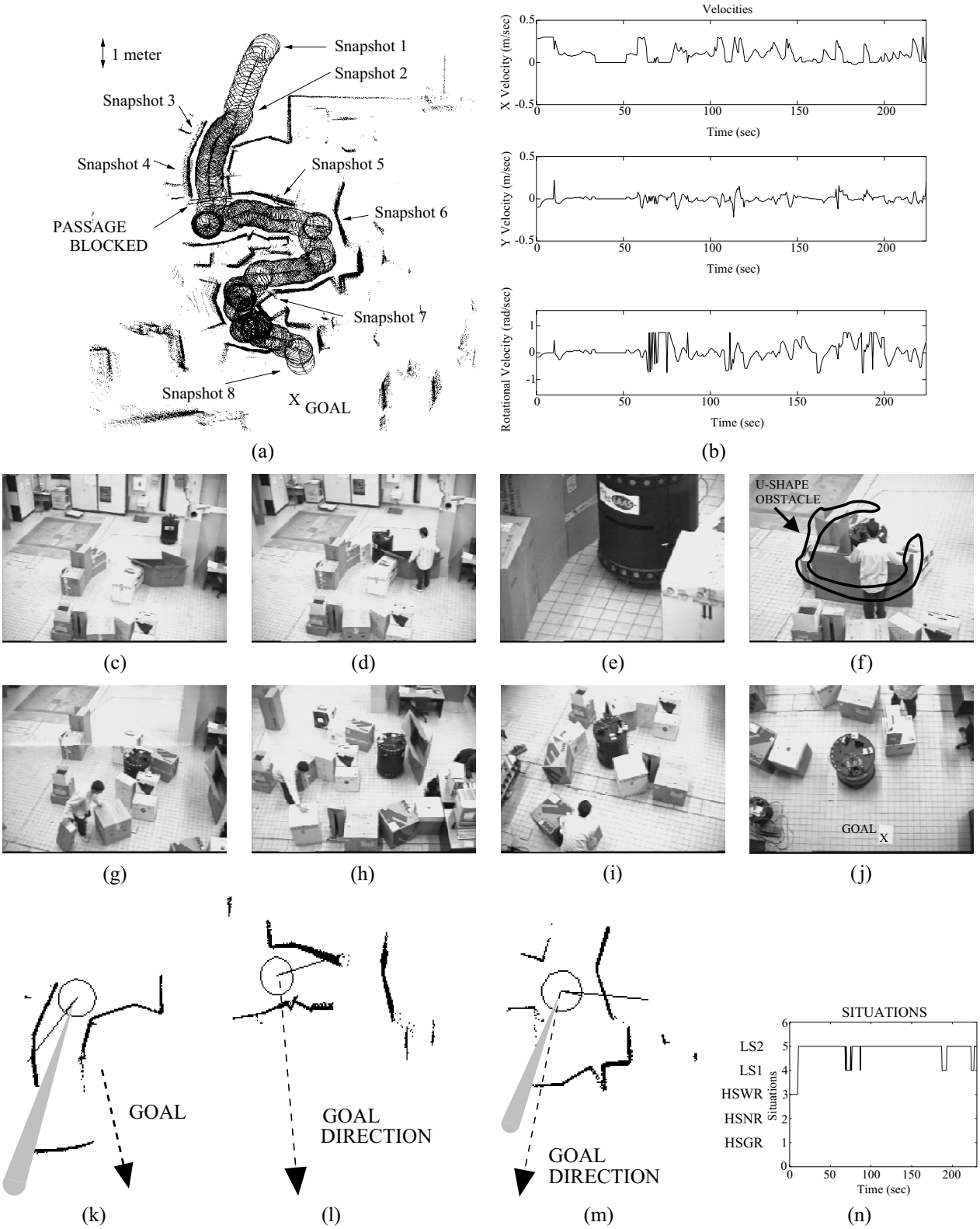

Fig. 7. Experiment 2.

method). There are methods based on a given path deformed in real time [6], [19], [20], [24]. A trap situation appears when the path lies within U-shaped obstacles dynamically created.

The ND method computes **oscillation-free motion** when the robot moves among very close obstacles, because the LS2 action was implemented to comply with this requirement [see the complete robot path and the velocity profile in Fig. 6(a) and (b) and Fig. 7(a) and (b)]. The potential field methods can produce oscillatory motion when moving among very close obstacles or narrow corridors [22].

Motion directions far away from the goal direction are obtained with the ND method (mentioned before as **goal insensi-**

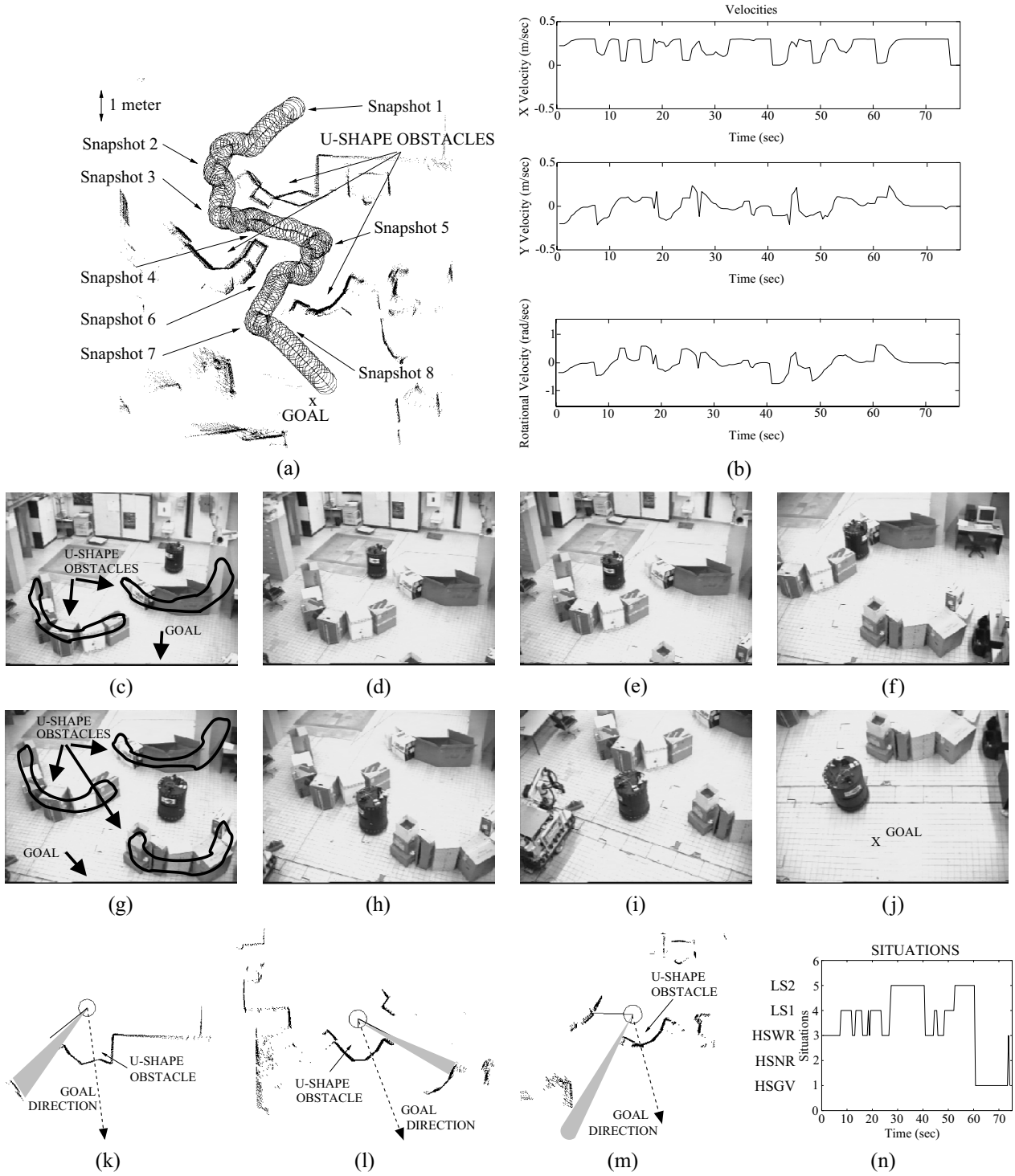

Fig. 8. Experiment 3.

tivity). This is because the goal direction is only used directly in one of the five motion laws (in HSGR, where the robot is not in danger and there is not an apparent navigation difficulty). This property was essential in many situations encountered in the experiments [see Fig. 7(l) and (m) and Fig. 8(m)]. The reactive methods that make a physical analogy use the goal location directly in the motion heuristic (e.g., [3]–[10]). These methods exhibit high goal sensitivity, so directions of motion far away from the goal location are difficult to obtain (in all the situations where they are required). In the methods that solve the problem with a constrained optimization (e.g., [11], [15], [16], [18]), one of the balance terms is the goal heading. Therefore, these methods also exhibit high goal sensitivity.

In the ND method, nothing prohibits the selection of **motion directions toward the obstacles**, so they are computed when required [see Fig. 6(k)–(m), Fig. 7(k)–(m), and Fig. 8(l) and (m)]. However, some methods explicitly prohibit the selection of motion toward the obstacles (e.g., [13]).

One difficulty found in most of the collision avoidance approaches is the **tuning of the internal parameters**. It is difficult to find the optimum values for a good behavior in all the collision avoidance situations. The ND method only has one parameter that is chosen heuristically (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. We implemented a hysteresis behavior to smooth transitions between some situations.

We have not addressed the sensor noise in the ND method implementation. We believe that external modules should process the sensory information in order to deal with noisy sensors (e.g., [25]). However, strategies such as increasing the security distance according to the sensor uncertainty could be designed.

Seen as a whole, the ND method is a robust reactive navigation method, which is mainly the result of two facts.

- Using a "divide and conquer" strategy to decompose the reactive navigation problem in subproblems (by different situations) and developing strategies for motion in any situation.
- Using the free walking area device gives the guarantee that it is possible to reach the goal, or that it is possible to reach the closest point to the goal within the maximum reach of the sensory information (the gap closest to the goal).

B. ND Navigation Limitations

We think that the main limitation of the ND method is the portability to different types of robots, because it does not take into account noncircular shapes or the vehicle kinematic and dynamic constraints. Some existing methods consider the robot shape (e.g., [3], [11], [16], [18]); others compute motion commands that comply with the robot kinematics (e.g., [11], [13], [15], [16], [18]) and others with the robot dynamics (e.g., [11], [15], [16], [18]).

It is difficult for the ND method to deal with noncircular shapes since it is formulated to apply over the workspace, while the classical space used to represent the robot geometry is the configuration space [26]. We have developed an underconstrained solution for square and rectangular shapes [27]. We have also proposed a spatial representation to deal with the kinematic constraints [28]. Using this work, the ND method can be used on two-wheeled, tricycle, and car-like robots. We also constructed a space to represent the vehicle dynamics [29]. From these results, the ND velocity limits can be significantly increased and safety is guaranteed.

C. 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 cannot guarantee global convergence to the goal location, because they use a local fraction of the information available (sensory information). Recently, some researchers have worked on introducing global information into the reactive methods to avoid the global trap situations. For example, [23] uses a look-ahead verification to analyze the consequences of heading toward the candidate directions, avoiding the trap situations by running the algorithm a few steps in advance of the algorithm execution. Furthermore, [20], [24], and [30] exploit the information of the connectivity of the space using a navigation function, which provides global information to the reactive method to avoid trap situations. In addition, these approaches are adapted to work in highly dynamic scenarios.

D. ND Navigation Background

We have validated the ND method on the Diligent[5] robot at LAAS-CNRS (Toulouse, France). Furthermore, this method was integrated as the low-level motion generator of the vehicle and it is used daily for demonstrations [31]. In the Robels system [32], 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, which are described in [30]).

With some modifications, the method works in other indoor/outdoor mobile platforms (see [27]: Hilare, Hilare2,[6] and Lama[7] at LAAS-CNRS (France); Otilio[8] at the University of Zaragoza (Spain), and r2[9] at the Technical University of Lisbon (Portugal). Currently, the method is being implemented on Dalay[10] at LAAS-CNRS (France).

VIII. Conclusions

We addressed in this paper reactive collision avoidance for mobile robots. We have presented the design of a reactive navigation method using the situated-activity paradigm of behavioral design. The advantage is that our design employs a "divide and conquer" strategy to reduce the difficulty of the navigation problem. As a consequence, the reactive navigation methods implemented (following the design guidelines) must be able to successfully navigate in more troublesome scenarios than other existing methods.

Our reactive method design has been used to implement some reactive navigation methods adapted to their collision avoidance context (for example, the free zone method [33] for soccer-player robots). We have used the design guidelines 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 other methods. Currently, the ND method is working as the reactive module in several robots at different laboratories. Although the method is presented here for circular and holonomic robots, it has been extended to work in vehicles with other shapes, and with kinematic and dynamic constraints.

[5]Diligent is a Nomadic XR4000 platform equipped with a 2-D planar laser.

[6]Hilare and Hilare2 are indoor rectangular differential-driven robots equipped with 2-D planar lasers.

[7]Lama is a rectangular outdoor robot that can work in differential-driven mode. The sensor used was a pair of black-and-white cameras.

[8]Otilio is square and differential-driven indoor robot equipped with a 3-D laser.

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

[10]Dalay is a rectangular and differential-driven outdoor robot equipped with a 2-D planar laser.

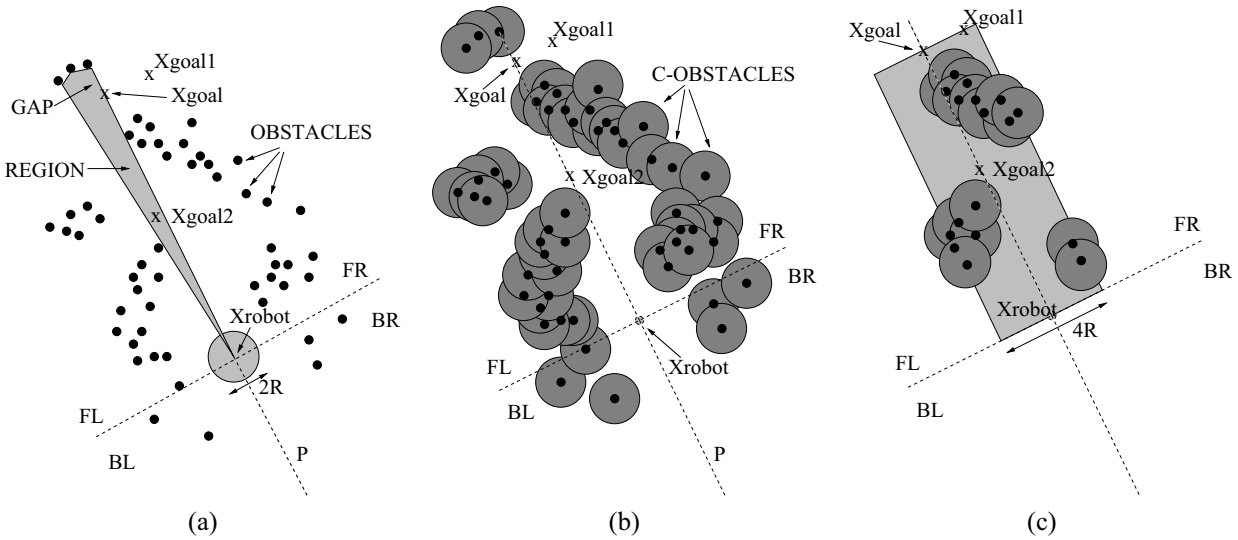

Fig. 9. Example of how the algorithm checks whether a point can be reached in the space.

APPENDIX

We introduce in this Appendix a procedure to verify whether a region is "navigable" for a circular and holonomic robot. First, we present an algorithm to check whether the robot can reach a location in the space, and next we use it to verify whether a region is "navigable."

A. The Basic Algorithm

The algorithm computes the existence of a path that connects the robot location and a point of the space (notice that the algorithm does not compute a path). The **inputs** of the algorithm are as follows.

1) The robot location $(\mathbf{x_{robot}})$ and robot radius (R).
2) The goal location $(\mathbf{x_{goal}})$.
3) A list (L) of obstacle points, where an obstacle is $(\mathbf{x_j^L})$.

The **output** of the algorithm is whether the goal location can be reached from the robot location or not.

We first divide the plane in four semiplanes (FL, FR, BL, BR)[11] by the line (named $\mathbf{P}$) that contains $\mathbf{x_{robot}}$ and $\mathbf{x_{goal}}$, and the perpendicular line to the previous line over $\mathbf{x_{robot}}$ [Fig. 9(a)]. Then:

1) if $\exists i / d(\mathbf{x_{goal}}, \mathbf{x_i^L}) < R$ then $\mathbf{x_{goal}}$ cannot be reached;
2) eliminate from $\mathbf{L}$ every point k that:
   (a) $\mathbf{x_k^L} \in BL$ or $\mathbf{x_k^L} \in BR$;
   (b) $d(\mathbf{x_k^L}, \mathbf{x_{robot}}) > d(\mathbf{x_{goal}}, \mathbf{x_{robot}})$;
   (c) $d(\mathbf{x_k^L}, \mathbf{P}) > 2R$;
3) if for all the remaining points of $\mathbf{L}$, $d(\mathbf{x_j^L}, \mathbf{x_k^L}) > 2R$ (with $\mathbf{x_j^L} \in FR$ and $\mathbf{x_k^L} \in FL$), then $\mathbf{x_{goal}}$ can be reached, else, it cannot be reached.

We discuss next the algorithm step by step.

1) The algorithm checks whether the $\mathbf{x_{goal}}$ is within any $\mathcal{C}$-obstacle[12] [26] to detect whether the goal location is in collision. We show the configuration space in Fig. 9(b), where $\mathbf{x_{goal}}$ is not within a $\mathcal{C}$-obstacle.
2) The algorithm eliminates the obstacle points that are out of the rectangle with height the segment that joins $\mathbf{x_{robot}}$ and $\mathbf{x_{goal}}$, and width $4R$ [Fig. 9(c)]. Within this rectangle is where the path is searched for.
3) The algorithm checks intersections among $\mathcal{C}$-obstacles of FR and FL. If there are no intersections, then there is a collision-free path that joins $\mathbf{x_{robot}}$ and $\mathbf{x_{goal}}$. Notice how in Fig. 9(c) there are not intersections among obstacles in FL and FR, thus there are many collision-free paths that join the robot and goal location within the rectangle.

The usefulness of this algorithm for reactive navigation is to know if a given point of the space can be reached (without explicitly computing any path). Then, this point could be a landmark point or the goal location itself. However, the algorithm could fail in some situations. For instance, if we use the algorithm with $\mathbf{x_{goal1}}$, the solution is that it cannot be reached while there exists a path. We show how to solve this situation in the next subsection.

B. Algorithm to Verify Whether a Region is "Navigable"

We use the algorithm to verify if a region is "navigable" as follows.

1) If the goal location is inside the region, then the algorithm checks if the goal location can be reached [e.g., $\mathbf{x_{goal2}}$ in Fig. 9(a)].
2) If the goal location is not inside the region, then the algorithm checks whether the middle point of the gap (closest to the goal location) of the region can be reached (e.g., if the goal location is $\mathbf{x_{goal1}}$ in Fig. 9(a), then we use the algorithm with $\mathbf{x_{goal}}$). Notice how the middle point of the gap $(\mathbf{x_{goal}})$ is used as a landmark in order to reach the goal location $(\mathbf{x_{goal1}})$.

In both cases, the solution of the algorithm is that both points can be reached. This region is "navigable" for both $\mathbf{x_{goal1}}$ and $\mathbf{x_{goal2}}$, and then it would be identified as the free walking area in both cases.

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

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

We remark that the case $\mathbf{x_{goal1}}$ is solved with the landmark $\mathbf{x_{goal}}$ computed from the region, but there are other cases that cannot be avoided by using local algorithms and require global information (see a discussion on this topic in Section VII-G).

ACKNOWLEDGMENT

The authors wish to thank all the members of R. Chatila's research group who hosted J. Minguez's visit to LAAS-CNRS. In particular, we thank R. Alami and T. Simeon for their valuable comments and discussions, and S. Fleury for her help with the algorithm implementation on the XR4000 Nomadic platform.

REFERENCES

[1] R. C. Arkin, *Behavior-Based Robotics*. Cambridge, MA: MIT Press, 1999.
[2] J. C. Latombe, *Robot Motion Planning*. Norwell, MA: Kluwer, 1991.
[3] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robot. Res.*, vol. 5, pp. 90–98, 1986.
[4] B. H. Krogh and C. E. Thorpe, "Integrated path planning and dynamic steering control for autonomous vehicles," in *Proc. IEEE Int. Conf. Robotics and Automation*, San Francisco, CA, 1986, pp. 1664–1669.
[5] R. B. Tilove, "Local obstacle avoidance for mobile robots based on the method of artificial potentials," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 2, Cincinnati, OH, 1990, pp. 566–571.
[6] M. Khatib, "Sensor-based motion control for mobile robots," Ph.D. dissertation, LAAS-CNRS, Toulouse, France, 1996.
[7] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Trans. Syst., Man, Cybern.*, vol. 19, pp. 1179–1187, May 1989.
[8] L. Montano and J. Asensio, "Real-time robot navigation in unstructured environments using a 3D laser rangefinder," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, vol. 2, Grenoble, France, 1997, pp. 526–532.
[9] K. Azarm and G. Schmidt, "Integrated mobile robot motion planning and execution in changing indoor environments," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, Munchen, Germany, 1994, pp. 298–305.
[10] A. Masoud, S. Masoud, and M. Bayoumi, "Robot navigation using a pressure generated mechanical stress field, the biharmonical potential approach," in *Proc. IEEE Int. Conf. Robotics and Automation*, San Diego, CA, 1994, pp. 124–129.
[11] W. Feiten, R. Bauer, and G. Lawitzky, "Robust obstacle avoidance in unknown and cramped environments," in *Proc. IEEE Int. Conf. Robotics and Automation*, San Diego, CA, 1994, pp. 2412–2417.
[12] J. Borenstein and Y. Koren, "The vector field histogram—fast obstacle avoidance for mobile robots," *IEEE Trans. Robot. Automat.*, vol. 7, pp. 278–288, Apr. 1991.
[13] I. Ulrich and J. Borenstein, "VFH+: reliable obstacle avoidance for fast mobile robots," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1998, pp. 1572–1577.
[14] M. Hebert, C. Thorpe, and A. Stentz, *Intelligent Unmanned Ground Vehicles: Autonomous Navigation Research at Carnegie Mellon*. Norwell, MA: Kluwer, 1997.
[15] R. Simmons, "The curvature-velocity method for local obstacle avoidance," in *Proc. IEEE Int. Conf. Robotics and Automation*, Minneapolis, MN, 1996, pp. 3375–3382.
[16] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robot. Automat. Mag.*, vol. 4, pp. 23–33, Mar. 1997.
[17] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *Proc. IEEE Int. Conf. Robotics and Automation*, Detroit, MI, 1999, pp. 341–346.
[18] K. Arras, J. Persson, N. Tomatis, and R. Siegwart, "Real-time obstacle avoidance for polygonal robots with a reduced dynamic window," in *Proc. IEEE Int. Conf. Robotics and Automation*, Washington, DC, 2002, pp. 3050–3055.
[19] S. Quinlan and O. Khatib, "Elastic bands: connecting path planning and control," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 2, Atlanta, GA, 1993, pp. 802–807.
[20] O. Brock and O. Khatib, "Real-time replanning in high-dimensional configuration spaces using sets of homotopic paths," in *Proc. IEEE Int. Conf. Robotics and Automation*, San Francisco, CA, 2000, pp. 550–555.
[21] M. Ginger, "Universal planning: an (almost) universally bad idea," *AI Mag.*, vol. 10, no. 4, pp. 40–44, Winter 1989.
[22] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 2, Sacramento, CA, 1991, pp. 1398–1404.
[23] I. Ulrich and J. Borenstein, "VFH*: local obstacle avoidance with look-ahead verification," in *Proc. IEEE Int. Conf. Robotics and Automation*, San Francisco, CA, 2000, pp. 2505–2511.
[24] O. Brock, "Generating robot motion: The integration of planning and execution," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1999.
[25] J. Borenstein and Y. Koren, "Histogramic in-motion mapping for mobile robot obstacle avoidance," *IEEE J. Robot. Automat.*, vol. 7, pp. 535–539, Aug. 1991.
[26] T. Lozano-Perez, "Spatial planning: a configuration space approach," *IEEE Trans. Comput.*, vol. C-32, pp. 108–120, Mar. 1983.
[27] J. Minguez and L. Montano, "Robot navigation in very complex dense and cluttered indoor/outdoor environments," presented at the *Proc. 15th IFAC World Congr.*, Barcelona, Spain, 2002.
[28] J. Minguez, L. Montano, and J. Santos-Victor, "Reactive collision avoidance for nonholonomic robots using the ego-kinematic space," in *Proc. IEEE Int. Conf. Robotics and Automation*, Washington, DC, 2002, pp. 3074–3080.
[29] J. Minguez, L. Montano, and O. Khatib, "Reactive collision avoidance for navigation at high speeds or systems with slow dynamics," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, Lausanne, Switzerland, 2002, pp. 588–594.
[30] J. Minguez, L. Montano, N. Simeon, and R. Alami, "Global nearness diagram navigation (GND)," in *Proc. IEEE Int. Conf. Robotics and Automation*, Seoul, Korea, 2001, pp. 33–39.
[31] R. Alami, I. Belousov, S. Fleury, M. Herb, F. Ingrand, J. Minguez, and B. Morisset, "Diligent: toward a human-friendly navigation system," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Takamatsu, Japan, 2000, pp. 2094–2100.
[32] B. Morisset and M. Gallab, "Learning how to combine sensory-motor modalities for a robust behavior," in *Advances in Plan-Based Control of Robotic Agents, Lecture Notes in Artificial Intelligence 2466*. New York: Springer, 2002, pp. 157–178.
[33] C. Marques and P. Lima, "Multi-sensor navigation for soccer robots," in *2001: Robot Soccer World Cup V*. Berlin, Germany: Springer-Verlag, July 2002.

Javier Minguez (S'00–A'02) received the physics science degree in 1996 from the Universidad Complutense de Madrid, Madrid, Spain, and the Ph.D. degree in computer science and systems engineering in 2002 from the University of Zaragoza, Zaragoza, Spain.

During his student period, in 1999 he was with the Robotics and Artificial Intelligence Group, LAAS-CNRS, Toulouse, France, for eight months. In 2000, he visited the Robot and Computer Vision Laboratory (ISR-IST), Technical University of Lisbon, Lisbon, Portugal, for ten months. In 2001, he was with the Robotics Laboratory, Stanford University, Stanford, CA, for five months. He is currently a full-time Researcher in the Robot, Vision, and Real Time Group, University of Zaragoza. His research interests are techniques for reactive navigation and sensor-based motion planning for mobile robots.

Luis Montano (M'01) was born on September 6, 1958 in Huesca, Spain. He received the industrial engineering degree in 1981 and the PhD degree in 1987, both from the University of Zaragoza, Zaragoza, Spain.

He is currently an Associate Professor of Systems Engineering and Automatic Control at the University of Zaragoza, and Principal Researcher in robotic research projects. He is also the coordinator of the Robotics, Perception, and Real Time group at the Aragon Institute of Engineering Research, University of Zaragoza. The group works in research and development projects within the areas of robotics, automation, and real time, and more specifically in task coordination, mobile robot navigation, simultaneous localization, and mapping (SLAM) and automatic 3-D model building. Previously, he was Head of the Computer Science and Systems Engineering Department, University of Zaragoza. His major research interests are robotics and perception systems.