Planning robust motion strategies for a mobile robot

Rachid Alami, Thierry Simeon

Planning robust motion strategies for a mobile robot

R. Alami and T. Siméon *
LAAS / CNRS
7, Avenue du Colonel Roche - 31077 Toulouse CEDEX (France)
e-mail: rachid@laas.fr, nic@laas.fr

Abstract

This paper reports on a recent work we have conducted concerning the development and the implementation of a robust motion planner for a mobile robot in a polygonal environment and in presence of uncertainty in robot control and sensing. Such a planner takes explicitly into account the uncertainty in robot control and produces a robust motion plan composed of sensor-based motion commands. The main originality of this approach is that it is able to account for a set of motion commands which may accumulate errors. It is based on a geometric analysis of the reachability and it generates motion strategies which may allow the robot to reduce its uncertainty.

1 Introduction

While the problem of planning collision-free motions has received considerable attention, there has been only a limited number of contributions where uncertainties are taken explicitly into account in the planning process. However, uncertainties arising from sensor noise, control error and inaccurate models, constitute an integral part of the planning problem for a real robot moving in a real environment.

Consequently, the use of motion planners which do not take explicitly into account uncertainty is limited to situations where the uncertainties can be dealt with by simply growing the obstacles. This cannot be the case in assembly tasks, or in situations where the initial uncertainty is greater than the task tolerances.

Other challenging problems arise for mobile robots which are not equipped with an absolute relocalization procedure and which may accumulate position errors. Indeed, in such situations, as the error cannot be bounded by a fixed value, a growing of the obstacles is not sufficient.

One may argue that uncertainty may be reduced by reading and interpreting sensory data at execution time, allowing to build a new plan after executing a portion of the original plan, etc... This is certainly insufficient when it is necessary to *reason in advance* , taking into account the capacity of the available sensing functions (visibility, range, specularity, uncertainty...), in order to generate a plan which allows to acquire the necessary knowledge to guide the robot towards the goal. Indeed, if the robot accumulates errors during its motion, it may even reach situations where it is "lost".

This paper reports on a recent work we have conducted concerning the development and the implementation of a robust motion planner for a mobile robot in a polygonal environment and in presence of uncertainty in control and sensing. Such a planner takes explicitly into account the uncertainty in robot control and produces a robust motion plan composed of sensor-based motion commands. The main originality of this approach is that it considers a set of motion commands which may accumulate errors. It is based on a geometric analysis of the reachability and it generates motion strategies which may allow the robot to reduce its uncertainty.

The paper is organized as follows. In section 2, we briefly discuss related work. Section 3 defines more precisely the problem we propose to tackle. In section 4 and 5, we describe how we perform a geometric analysis of the reachability taking into account the uncertainties, and how this analysis is used by the planner (section 6). We finally present some experimental results and discuss possible extensions.

2 Background and Related work

We do not have here the ambition to provide a detailed analysis of the relevant literature. The reader may refer to a very interesting state of the art and discussion in [11, 12, 9].

Roughly speaking, the problem has been tackled in the literature following two general approaches: several contributions are based on a *two-phase* approach, while other contributions stem from the *preimage backchaining* approach.

A/ In the *two-phase approach*, a motion plan (or more generally a task plan) is first generated assuming no uncertainty and then, this plan is analyzed and patched in order to finally produce a robust task plan. The analysis is mainly based on the propagation of uncertainty through the plan and its consequences on plan correctness [21, 14, 2, 15, 19, 22, ?, 18, 10]. The plan is then modified by inserting complementary actions or sensor readings allowing to reduce uncertainty.

*this work is supported by the CEC Esprit 3 Project 6546 PROMotion

When a plan cannot be patched, a backtrack can occur when planning is based on *skeleton* or *script* selection [15]. The main interest of such an approach is that it can applied to problems which can be more general than motion planning (e.g. assembly, manipulation) and to robots with a high number of degrees of freedom. Its main drawback is that it is based on an *a priori* decomposition of the planning process and on the assumption that, in most situations, the plan can be locally patched.

B/ The second approach is the *preimage backchaining approach* which is based on the geometry of the Configuration Space. It has been originally proposed by [16] and analyzed or extended through several contributions [7, 5, 4, 11, 12]. The underlying assumption is that it is necessary to take uncertainty explicitly into account in the planning process itself since it can have drastic consequences on the plan structure.

A *preimage* for a given motion command and for a given goal region (subset of the C-space), is a set of free configurations from which the command can be started and which guarantee that the robot will reach the goal region and stop inside it. Given a goal region and a start region,*preimage backchaining* consists in finding a sequence of commands (i.e. a plan) such that the inverse sequence allows an iterative construction of preimages starting from the goal and resulting in a preimage which contains the initial region.

While this problem, in its general formulation gives a useful computational framework, it raises "discouraging" complexity issues [4]. However this, by no means, affects its interest. It remains to find relevant instances of the general problem:

- which integrate new sensor modalities (proximity sensors, vision) and control algorithms [3, 8];
- which provide sub-classes of the problem with "better" properties [13];
- or which allow to implement algorithms of practical use even though they are not complete.

The problem that we have defined and the algorithm that we propose to solve it fall in these categories.

Note that we also need more elaborate models of different sensor modalities in order to be able to compute where (in the C-Space) they can be used and the nature of the measure they can provide together with its (pre-computed) estimated uncertainty. This should of course be done, taking into account the sensor characteristics (position relative to the robot frame, range, specularity...) in a given environment [6, 20].

3 Problem statement

3.1 The robot and the environment

Let us assume that the robot is a point moving in the plane amidst polygonal obstacles denoted by a set of oriented edges $\mathbf{e}_i$. We assume that the environment is perfectly known.

The robot is equipped with a position sensor (dead reckoning), and a proximity sensor which allows to detect contact with obstacles and to follow walls (edges). The error on position depends on past history (trajectory) and results in cumulative uncertainty.

We assume that the robot initial and final positions are in the free space or along a known edge with an associated uncertainty represented either by a disk or by an interval. The planner will provide a plan which allows, at any moment to know if the robot is at an uncertain position $\mathbf{p}$:

- in the free space; $\mathbf{p} = (x, y, \epsilon)$, where x, y denote the coordinates and ϵ the value of the maximum error of the position sensor (dead reckoning)[1];
- in the contact space, on a known edge $\mathbf{e}_i$; $\mathbf{p} = (x, \epsilon)$ where x denotes the nominal position along $\mathbf{e}_i$ and ϵ the value of maximum error;
- or on a known obstacle vertex; the uncertainty is then set to zero[2].

3.2 The commands and their models

There are several available motion primitives which are modelled by the type of initial situations where they can be applied, the final situation they can reach and the envelope of possible actual trajectories they can induce (Fig. 1):

- `MOVE_DISTANCE`(θ, d)
- `MOVE_UNTIL_CONTACT`(θ)
- `FOLLOW_WALL_DISTANCE`$(d,$ `[LEFT | RIGHT]`$)$
- `FOLLOW_WALL_UNTIL_VERTEX(`[LEFT | RIGHT]`)`

These primitives are based on dead-reckoning and contact (or proximity) sensors for primitives designed to maintain contact (or fixed distance) with an edge.

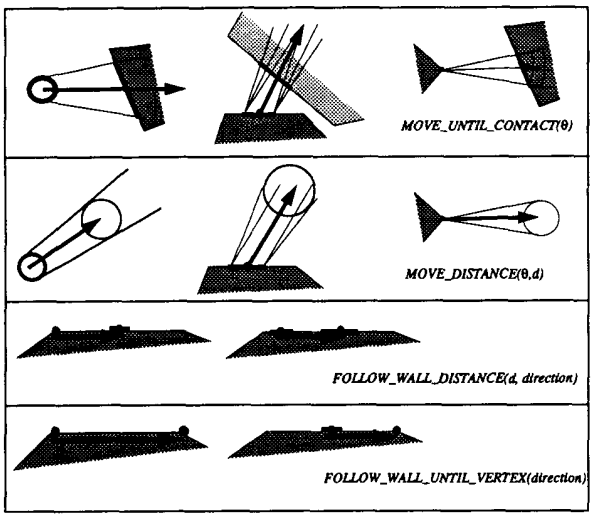

Figure 1: Model of the different primitives

1/ When the robot is commanded to move in a given direction θ (`MOVE_DISTANCE`, `MOVE_UNTIL_CONTACT`), the robot follows a path such that the tangent to the

[1] the disk centered on (x, y) with a radius equal to ϵ may overlap obstacles

[2] it is straightforward to extend it to a fixed value

path at any point makes an angle with the direction θ which is smaller than a pre-specified value $\Delta\theta$ called the *angle of the control uncertainty cone.*

2/ When the robot reaches without ambiguity an edge (after a **MOVE_UNTIL_CONTACT**), the new uncertainty is modelled as an interval whose endpoints correspond to the intersection of the edge with the envelope of all possible trajectories.

3/ When the robot is commanded to follow an edge (**FOLLOW_WALL_DISTANCE**), we assume that the maximum value of the positioning error, increases linearly: $\epsilon(x) = a.\|x - x_0\| + \epsilon_0$ with $a \in [0, 1]$ (Fig. 2-a).

4/ When the robot reaches a vertex (after a **FOLLOW_WALL_UNTIL_VERTEX**), ϵ is reset to zero. If, from the reached vertex, the robot follows again the same edge (in the opposite direction) or an adjacent edge, using **FOLLOW_WALL_DISTANCE**, the maximum value of the positioning error will again increase linearly, but with an initial value equal to zero: $\epsilon(x) = a.x$ (Fig 2-b and 2-c).

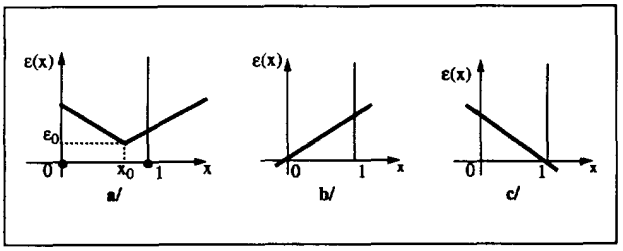

Figure 2: Evolution of the uncertainty along an edge

This will allow the planner to produce, when necessary, re-localization strategies along a given edge. In order to ensure a coherence between the different commands, we take: $a = \tan(\Delta\theta)$. However, the algorithm we propose does need such an assumption.

3.3 The planning problem

Given an initial and a goal position, in the free space or along a known edge together with their uncertainties (represented by a disk or by an interval), the planner should generate a sequence of motion commands whose execution guarantees to reach the goal with the desired uncertainty if the actual errors lie inside the control uncertainty cone $\Delta\theta$.

We restrict the plan, except the initial and final steps, to contain a sequence of commands which allow the robot to follow edges or to reach an edge starting from another one.

In order to implement such a planner, we need a basic step which computes if a given entity (a disk or an edge) can be reached from another entity using the available commands. If a command exists, the entities will be said *adjacent.*

Informally, a target entity ent_j can be reached from an uncertain position **p**, if there exists at least one direction θ such that, taking into account a given value of the uncertainty cone $\Delta\theta$:

- **C1:** the robot cannot miss ent_j (i.e. all possible paths inside the uncertainty cone, for all possible initial configurations, intersect ent_j); ent_j is said to be *Strongly visible* from **p**;
- **C2:** the robot cannot collide any other entity ent_k (i.e. no possible path inside the uncertainty cone, for all possible initial configurations intersect another entity before its intersection with ent_j); if it collides an entity ent_k, ent_k is said to be *Weakly visible* from **p**;

We develop, in the next sections, the computation of the disk-edge adjacency and edge-edge adjacency. While the first is quite simple (§4), the second gives rise to the possibility of characterizing the adjacency from a *complete* edge and not only from an uncertain position on an edge (§5). This is possible because we have a knowledge on the evolution of uncertainty along an edge. It will provide us with a coherent way to combine and to compute the parameters for **MOVE_UNTIL_CONTACT**, **FOLLOW_WALL_DISTANCE** and **FOLLOW_WALL_UNTIL_VERTEX** commands.

4 Disk-Edge Adjacency

This adjacency can be easily computed as illustrated by Fig. 3. The strong visibility is satisfied from at most one range of θ values, while the weak visibility can correspond to one range of θ values for each entity in the environment different from the target entity. The adjacency will be then verified for a set of θ ranges corresponding to the set difference between the strong visibility range and the weak visibility ranges. The θ-ranges are determined using the tangents to the disk going through the edges endpoints. Figure 4 shows an example of workspace and the adjacencies computed by the algorithm.

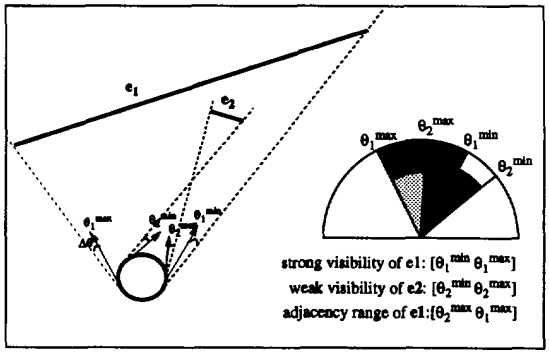

Figure 3: θ-ranges associated to the disk-edge adjacency

5 Edge-Edge adjacency

Let us consider that the robot arrived on a given edge $\mathbf{e}_i$ at position (x_0, ϵ_0). We want to characterize the parameters of the motion commands which allow to reliably reach the other edges of the environment from this position. Some of these edges can be directly

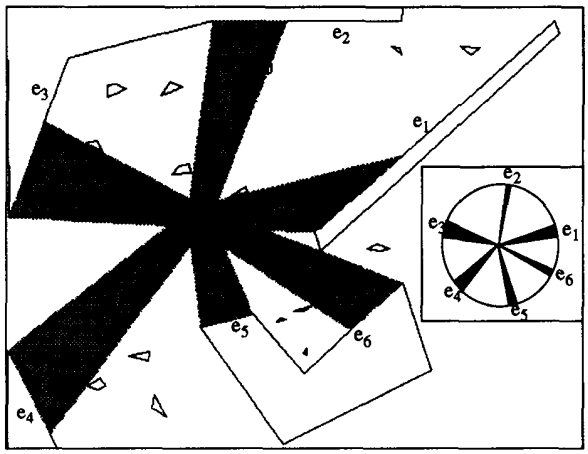

Figure 4: An example of disk/edge adjacency

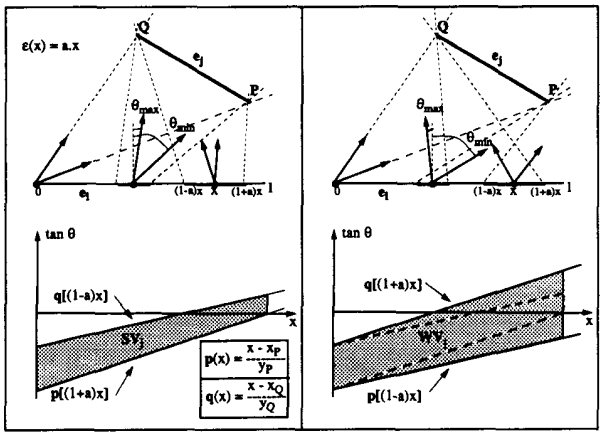

Figure 5: Strong and Weak visibility regions

reached with a **MOVE_UNTIL_CONTACT** using a computation similar to the case described in §4. However there can also be other edges which are not directly "visible" from the current position but which can be reached if the robot first executes a **FOLLOW_WALL_DISTANCE** along e_i from x_0 to another position x. In some other cases, the additional uncertainty induced by **FOLLOW_WALL_DISTANCE** will be too important and a re-localization (using **FOLLOW_UNTIL_VERTEX**) will be required. In this section, we first introduce a notion of visibility between edges; then we explain how this notion is used to decompose the (x, θ)-space (which parametrizes the set of motion commands) into regions where the reachability of specific edges is guaranteed. These regions allow to determine the adjacencies between the edges of the environment.

5.1 Visibility regions

An edge e_j is said to be *strongly* (resp. *weakly*) visible from a position x along e_i and for a commanded motion in direction θ, if for **any** (resp. **some**) uncertain position $\bar{x} \in [x - \epsilon(x), x + \epsilon(x)]$, the straight line issued from $\bar{x}$ in direction θ intersects the edge e_j. The regions of the (x, θ)-space which verify this definition.are called the **strong** (resp. **weak**) **visibility** regions from e_i to e_j; they are respectively denoted $\mathcal{SV}_j$ and $\mathcal{WV}_j$.

These regions are limited by two curves $\theta_{min}(x)$ and $\theta_{max}(x)$. For a linear evolution of the uncertainty $\epsilon(x)$, it can be easily shown that the tangents of these extremal orientations also depend linearly on x. Thus, $\mathcal{SV}_j$ and $\mathcal{WV}_j$ can be simply represented in the $(x, \tan(\theta))$-space by trapezoidal regions. Figure 5 shows an example which illustrates the construction of the visibility regions.

Note that more complex situations occur when the start edge is intersected by the support line of e_j. In this case, $\mathcal{SV}_j$ and $\mathcal{WV}_j$ may possibly consist of several trapezoidal subregions [3].

5.2 Adjacency regions

The **adjacency regions** $\mathcal{A}_j$ from e_i to e_j, consist of the (x, θ) values such that the motions issued from any uncertain position $x \pm \epsilon(x)$ along e_i, and in any direction $\theta \pm \Delta\theta$ are guaranteed to endup on e_j; that is, the robot will not miss e_j and it will not be stopped on its way by another edge. We describe below the two steps of the algorithm we implemented in order to compute the $\mathcal{A}_j$ regions.

The first step is performed without considering the directional uncertainty $\Delta\theta$. In this case, $\mathcal{A}_j$ clearly corresponds to the sub-regions of $\mathcal{SV}_j$ which are not contained into the union of the $\mathcal{WV}_k$ regions associated to the edges lying between e_i and e_j. $\mathcal{A}_j$ is first initialized to $\mathcal{SV}_j$; then it is iteratively decomposed by considering each of the $\mathcal{WV}_k$. The basic operation of each iteration consists in computing the set difference between two regions. This computation can be easily performed by considering their representations in the $(x, \tan(\theta))$-space; it results in decomposing each trapezoid into at most four trapezoidal regions.

The second step is needed to account for $\Delta\theta$. Hence, the adjacency regions $\mathcal{A}_j$ only contain the points (x, θ) such that $(x, \tan(\theta \pm \Delta\theta)$ belongs to the trapezoids produced by the first step. Using the equations of the upper and lower lines defining each trapezoid, we derive the expression of the curves $\theta_{min}(x)$ and $\theta_{max}(x)$ which limit the corresponding sub-region of $\mathcal{A}_j$.

Figure 6-b shows a simple environment and the adjacency regions $\mathcal{A}_7$ computed between the edges e_1 and e_7. The external contours correspond to the boundary of the adjacency regions before "shrinking" by $\Delta\theta$.

[3] at most two disconnected regions for $\mathcal{SV}_j$ and three connected regions for $\mathcal{WV}_j$. See [1] for a more detailed analysis.

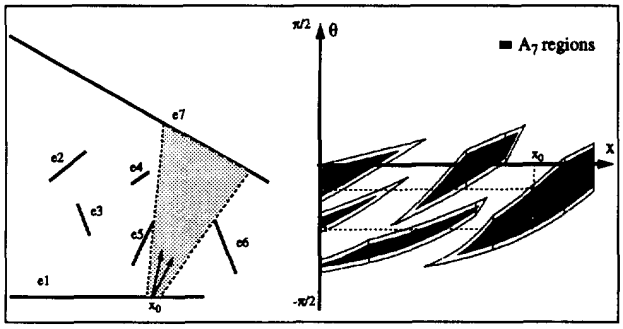

Figure 6: Example of adjacency regions from e_1 to e_7

6 The Planner

The planner is given a set of polygonal obstacles, a value for the angle uncertainty cone $\Delta\theta$, the initial and final positions represented by two discs C_I and C_G. The search space is a directed graph G of uncertain positions $\mathbf{p} = (x, y, \epsilon)$. The nodes are the initial and final positions $\mathbf{p}_I$ and $\mathbf{p}_G$ and a discrete set of positions lying on the edges of the environment. An arc between two nodes correspond to a given motion command as defined in §3.2.

The planner initially constructs a graph G containing only the two nodes associated to $\mathbf{p}_I$ and $\mathbf{p}_G$. In a first step, it simply tries to connect them by checking that the half-lines D_1 and D_2 issued from $\mathbf{p}_I$ (Fig. 7) do not intersect any edge $\mathbf{e}_i$ and both intersect the circular region C_G. In this case, the single motion command **MOVE_DISTANCE**(θ_{IG}, d_{IG}) issued in direction θ_{IG} and terminating after a nominal distance d_{IG} is guaranteed to end somewhere inside the goal region.

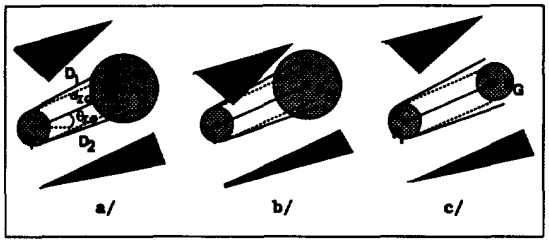

Figure 7: a/ $\mathbf{p}_G$ and $\mathbf{p}_I$ are connected. b/ and c/ $\mathbf{p}_G$ and $\mathbf{p}_I$ cannot be connected (b/ potential collision with the environment, c/ final error too large)

If not, the initial node is expanded in order to generate a discrete set of positions lying on the "visible" edges of the environment. These successors are created for the extremal values of the legal θ-ranges. The arcs issued from $\mathbf{p}_I$ correspond to a **MOVE_UNTIL_CONTACT** motion command.

The other nodes of the graph are incrementally created during the search of a minimum-cost path connecting $\mathbf{p}_I$ to $\mathbf{p}_G$. The costs assigned to the arcs are simply computed from the length of the nominal path connecting two adjacent nodes, and the search is carried out by a classical A^* algorithm. Each iteration of the algorithm consists in expanding the "best" node of the *OPEN* list. We now describe the expansion of a current node denoted $\mathbf{p}_{cur}$ lying on an edge $\mathbf{e}_{cur}$. If $\mathbf{p}_{cur}$ can be directly connected to $\mathbf{p}_G$, the expansion simply consists in linking both nodes with a **MOVE_DISTANCE** arc and the planner returns success. Otherwise, the successors are generated as follows: p_{cur} is first connected with **FOLLOW_UNTIL_VERTEX** motions to the two endpoints of edge $\mathbf{e}_{cur}$. As the robot is assumed to perfectly detect a corner of the environment, a null uncertainty is associated to both corresponding nodes (p_{left} and p_{right} for the example of Fig. 8 which illustrates the node expansion mechanism).

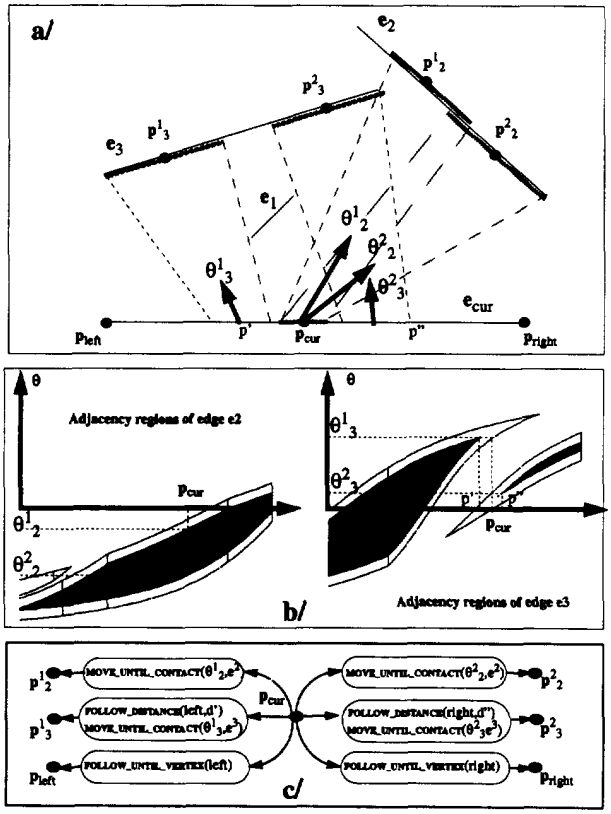

Figure 8: Node expansion mechanism

For all the other edges $\mathbf{e}_i$, the adjacency regions computed from $\mathbf{e}_{cur}$ to $\mathbf{e}_i$ are used to generate motion commands guaranteed to reach the edge $\mathbf{e}_i$. Three cases can occur:

- The set of adjacency regions $\mathcal{A}_i$ is empty (ie. the edge is too short to be reachable from $\mathbf{e}_{cur}$ with the current uncertainty or it is completely obstructed by other edges): no successors are associated to this edge. This is the case of edge $\mathbf{e}_1$ on the example.
- The location $\mathbf{p}_{cur}$ along the edge $\mathbf{e}_{cur}$ is such that one (or more) feasible range of orientations exists in

the $\mathcal{A}_i$'s regions (case illustrated by the $\mathcal{A}_2$ region shown in Fig. 8-b). A successor is then created for each of the two extremal orientations of the ranges (θ_2^1 and θ_2^2 for the example). These new nodes are connected to $\mathbf{p}_{cur}$ by **MOVE_UNTIL_CONTACT** commands.

• When the vertical line issued in the (x, θ)-space at the current x position does not intersect any cells of the $\mathcal{A}_i$ regions (case illustrated by the $\mathcal{A}_3$ regions of Fig. 8-b associated to the edge $\mathbf{e}_3$), the robot first needs to follow $\mathbf{e}_{cur}$ toward the left (resp. the right) until it reaches a position denoted $\mathbf{p}'$ (resp. $\mathbf{p}''$) on the figure, and corresponding to the beginning of the nearest legal range. From this intermediate position $\mathbf{p}'$ (resp. $\mathbf{p}''$) reached via a **FOLLOW_DISTANCE** command, a **MOVE_UNTIL_CONTACT** motion command issued in direction θ_3^1 (resp. θ_3^2) allows to generate the successor $\mathbf{p}_3^1$ (resp. $\mathbf{p}_3^2$) on edge $\mathbf{e}_3$.

Figure 8-c shows the successors created for the environment of Fig. 8-a; it also indicates the motion commands produced to connect $\mathbf{p}_{cur}$ to these nodes.

7 Experimental Results

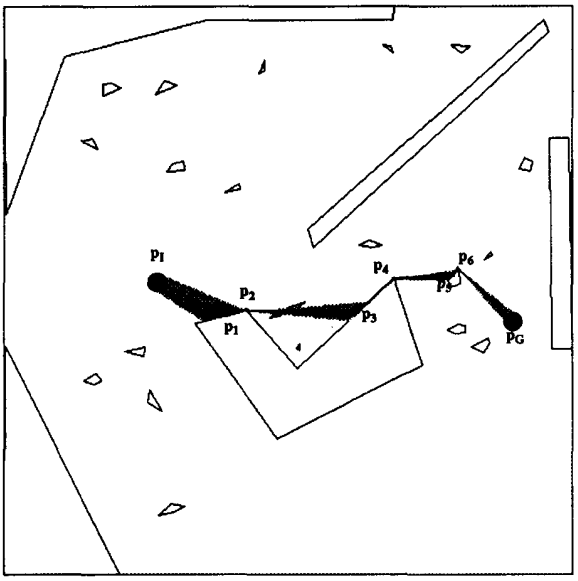

Figure 9: Solution obtained for small initial and control uncertainty

The algorithms presented above have been implemented in C on a Sun Sparc 10 computer. We ran the planner on several workspaces, with various sizes of the initial and goal regions (ϵ_I and ϵ_G) and with different values of the directional uncertainty $\Delta\theta$.

Figures 9, 10 and 11 illustrate some results obtained for a same polygonal workspace containing 98 edges. The black polyline joining the initial and goal regions represent the nominal trajectory and the grey regions show the areas that may by swept by the robot during the execution of the motion commands because of the uncertainties.

The first example (Fig. 9) was run for small values of both initial and control uncertainties ($\Delta\theta = 5deg.$). Note that the positional uncertainty is reset at the end of the **FOLLOW_UNTIL_VERTEX** commands (positions $\mathbf{p}_2$, $\mathbf{p}_4$ and $\mathbf{p}_6$) because of the vertex based re-localization. Only 35 nodes were developed during the search and the solution was obtained in half a second after the computation of the adjacency regions between the edges which required 6 seconds on a Sparc 10 workstation.

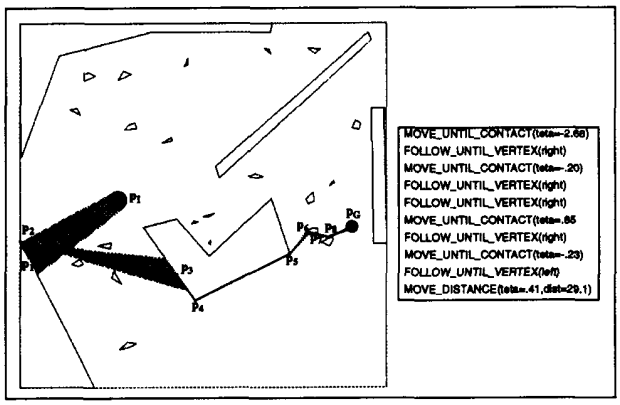

Figure 10: Same example with a larger initial uncertainty

The second example was run with the same value of $\Delta\theta$ but with a larger radius for the initial disk. In this case, the initial uncertainty is too important to directly reach the polygon displayed at the center of the figure. After a first motion guaranteed to stop on the left-down wall of the workspace, and a precise vertex localization at position $\mathbf{p}_2$, this large polygon can be reliably reached with the next **MOVE_UNTIL_CONTACT** command (position $\mathbf{p}_3$).

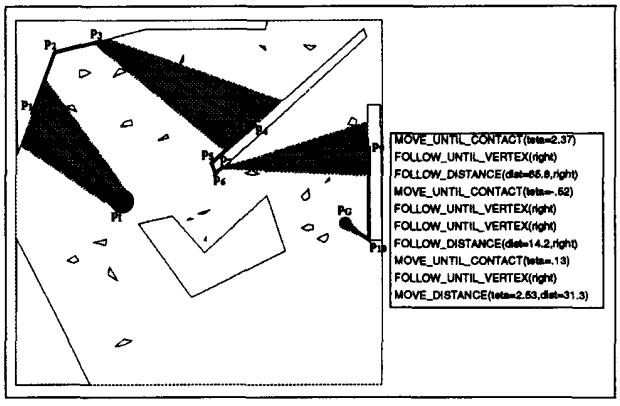

Figure 11: Same example with a larger control uncertainty

Finally, the third example was run with $\Delta\theta$ set to 10 degrees. The small obstacles around the goal region

cannot be used anymore to localize the robot and a completely different motion strategy is required as illustrated by Fig. 11. Note that the solution produced by the planner contains two **FOLLOW_DISTANCE** commands between p_2 (resp. p_6) and p_3 (resp. p_7). In both cases, the edge containing p_4 (resp. p_9) could not be reached directly from position p_2 (resp. p_6) and the planner used the adjacency regions between the corresponding edges to compute the first position at which the next **MOVE_UNTIL_CONTACT** command was guaranteed to succeed.

For the two last examples, 46 and 78 nodes were respectively developed during the search and the solution was obtained in both cases after less than one second of computation.

8 Discussion and Extensions

The main originality of the approach described above is its ability to produce robust motion plans composed of sensor-based motion commands which may accumulate errors, and to determine where the robot needs to re-localize in order to guarantee a successful execution of the trajectory.

Although the edge-edge adjacency described in § 5 captures exactly all the guaranteed motions between two given edges of the workspace, the planner described in § 6 is not complete since the number of paths explored is a finite set (discrete points (x, θ) selected into the computed adjacency regions). However, the planner becomes complete if we restrict the class of solutions to strategies where a vertex re-localization is systematically performed after the **MOVE_UNTIL_CONTACT** commands (the errors do not depend anymore on the complete past history). Note that in this case, and when the error $\Delta\theta$ tends to zero, the search space of the algorithm degenerates to a classical visibility graph.

The algorithm described in this paper solves a relatively simple instance of the motion planning problem in presence of uncertainty. We are currently investigating several extensions of the approach. Some of them are rather straightforward; the algorithms can be easily adapted to deal with a circular robot and to relax the assumption of a perfect proximity sensor. Another interesting extension would be to consider the information which can be provided by other types of sensors (distance to the obstacle, orientation of the edges) and to explore the possibility of producing conditional strategies.

Nevertheless, there are numerous situations where it should be enough to use a planner like the one we described, which is really able to find in a very reasonable time, non-trivial sensor-based motions strategies for a mobile robot equipped with proximity sensors.

References

[1] R. Alami and T. Siméon. Planning robust motion stategies for a mobile robot. *Technical Report LAAS-93005*, October 93.

[2] R.A Brooks. Symbolic error analysis and robot planning. *Int. Journal of Robotics Research*, 1(4), 1982.

[3] R. Brost. Dynamic analysis of planar manipulation tasks. In *IEEE ICRA'92*, Nice (France).

[4] J. Canny. On computability of fine motion plans. In *IEEE ICRA'89*, Scottsdale (USA).

[5] B.R. Donald. A geometric approach to error detection and recovery for robot motion planning with uncertainty. *Artificial Intelligence*, 37:223–271, 1988.

[6] B.R. Donald, J. Jennings. Sensor interpretation and task-directed planning using perceptual equivalence classes. TR 91-1248, Cornell University, Dec 1991.

[7] M. Erdmann. Using backprojections for fine motion planning with uncertainty. *Int. Journal for Robotics Research*, 5(1):19–45, Spring 1986.

[8] A. Fox, S. Hutchinson. Exploiting visual constraints in the synthesis of uncertainty-tolerant motion plans (parts I & II) In *IEEE ICRA'93*, Atlanta.

[9] K. Goldberg, M. Mason, A. Requicha. Geometric uncertainty in motion planning: Summary report and bibliography. Technical report IRIS TR 297, USC Los Angeles, Aug 1992.

[10] S. N. Gottschlich, A. C. Kak. AMP-CAD: An assembly motion planning system. In *IEEE ICRA'92*, Nice.

[11] J.-C. Latombe. Robot Motion Planning. *Kluwer Academic Publishers*, 1991.

[12] J-C. Latombe, A. Lazanas, S. Shekhar. Robot motion planning with uncertainty in control and sensing. *Artificial Intelligence*, 52(1):1–47, Nov 1991.

[13] A. Lazanas, J-C. Latombe. Landmark-based robot navigation. Technical Report STAN-CS-92-1428, Stanford University, May 1992.

[14] T. Lozano-Perez. The design of a mechanical assembly system. A.I Memo 397, MIT Artificial Intelligence Lab., MIT, Cambridge, Dec 1976.

[15] T. Lozano-Perez. Robot programming. A.I Memo 698, MIT Artificial Intelligence Lab, Dec 1982.

[16] T. Lozano-Perez, M.T. Mason, R.H. Taylor. Automatic synthesis of fine motion strategies for robots. *Int. Journal of Robotics Research*, 3(1), 1984.

[17] I. Mazon, R. Alami. Representation and propagation of positionning uncertainties through manipulation robot programs.. In *IEEE ICRA'89* , Scottsdale.

[18] I. Mazon, R. Alami, P. Violéro. Automatic planning of pick and place operations in presence of uncertainties. In *IROS'90*, Tsuchiura (Japan), July 1990.

[19] J. Troccaz, P. Puget. Dealing with uncertainty in robot planning using program proving techniques. *The Fourt ISRR*, Santa Cruz (USA), March 1987.

[20] H. Takeda, J-C. Latombe. Sensory uncertainty field for robot navigation. In *IEEE ICRA'92*, Nice.

[21] R.H. Taylor. Synthesis of manipulator control programs from task-level specifications. AIM 228, AI Laboratory, Stanford, July 1976.

[22] R.H. Taylor, V.T. Rajan. The Efficient Computation of Uncertainty Spaces for Sensor-based Robot Planning. In *IEEE IROS'88*, Tokyo (Japan)