May 1992

Report No. STAN-CS-92-1428

PB96-152657

Landmark-Based Robot Navigation

by

Anthony Lazanas and Jean-Claude Latombe

Department of Computer Science

Stanford University

Stanford, California 94305

LANDMARK-BASED ROBOT NAVIGATION

ANTHONY LAZANAS
lazanas@flamingo.stanford.edu

JEAN-CLAUDE LATOMBE
latombe@cs.stanford.edu

ROBOTICS LABORATORY
DEPARTMENT OF COMPUTER SCIENCE
STANFORD UNIVERSITY, STANFORD, CA 94305, USA

Abstract

Achieving goals despite uncertainty in control and sensing may require robots to perform complicated motion planning and execution monitoring. This paper describes a reduced version of the general planning problem in the presence of uncertainty and a complete polynomial algorithm solving it. The planner computes a guaranteed plan (for given uncertainty bounds) by backchaining non-directional preimages of the goal until one fully contains the set of possible initial positions of the robot. The planner assumes that landmarks are scattered across the workspace, that robot control and sensing are perfect within the fields of influence of these landmarks, and that control is imperfect and sensing null outside these fields. The polynomiality and completeness of the algorithm derive from these simplifying assumptions, whose satisfaction may require the robot and/or its workspace to be specifically engineered. This leads us to view robot/workspace engineering as a means to make planning problems tractable. A computer program embedding the planner was implemented, along with navigation techniques and a robot simulator. Several examples run with this program are presented in this paper. Non-implemented extensions of the planner are also discussed.

Acknowledgments: This research was partially funded by DARPA contract DAAA21-89-C0002 (US Army).

Contents

1. INTRODUCTION

To operate in the real world robots must deal with errors in control and sensing. Achieving goals despite these errors requires complicated motion planning and execution monitoring [22]. Several approaches have been proposed to plan motion strategies in the presence of uncertainty, but many of them are based on unclear assumptions and/or are incomplete, yielding systems which are difficult to assess and to work from. The most rigorous approach so far has been the LMT approach (preimage backchaining) [26]. Several effective planning methods based on this approach have been proposed, but most of them require exponential time in the size of the input problem [12, 7], or they are incomplete with respect to the class of problems they attack [23]. Motion planning algorithms will not be applicable to real-world problems if they remain exponential or unreliable. Since the general problem seems to be intrinsically hard [5], a promising line of research is to identify a restricted, but still interesting subclass of problems that can be solved in polynomial time. This subclass can be obtained, for example, by engineering the robot and its workspace. *Thus, for the first time, engineering is formally seen as a means to make planning problems tractable.* Of course, robot/workspace engineering has its own cost, and we should be careful not to over-specialize the class of problems.

In this paper we consider a class of planning problems in the context of the navigation of a mobile robot. We assume that *landmarks* are scattered across the robot's two-dimensional workspace. Each landmark is a physical feature of the workspace, or a combination of features, that the robot can sense and identify if it is located in some appropriate subset of the workspace. This subset is the *field of influence* of the landmark. A landmark may be a pre-existing feature (e.g., the corner made by two walls) or an artificial one specifically provided to help robot navigation (e.g., a radio beacon or a magnetic device buried in the ground). We assume that robot control and sensing are perfect in the fields of influence associated with the landmarks, and that control is imperfect and sensing is null outside any such field. Given an initial region in the workspace, where the robot is known to be, and a goal region, where we would like the robot to go, the planning problem is to generate motion commands whose execution guarantees that the robot will move into the goal and stop there if our assumptions are satisfied. The motion commands should also prevent the robot from colliding with the obstacles.

We propose a planning method based on the LMT approach to solve the above problem. The method iteratively *backchains non-directional preimages* (weakest preconditions) of the goal, until one preimage encloses the set of possible initial positions of the robot. Each non-directional preimage is computed as a set of directional preimages for critical directions of motion. At every iteration, the intersection of the current non-directional preimage with the fields of influence of the landmarks

define the intermediate goal from which to backchain. The overall algorithm takes polynomial time in the total number of landmarks. It is complete with respect to the problems it attacks, that is, it produces a *guaranteed* plan (for input control uncertainty bounds), whenever one such plan exists, and returns failure, otherwise. (A guaranteed plan is one whose execution is guaranteed to succeed if the actual errors lie within the uncertainty bounds.) The polynomiality and completeness of the algorithm essentially derive from the combination of the two notions of a landmark and a non-directional preimage. An interesting aspect of the method is that, once a motion plan has been generated, the assumption that control and sensing are perfect in landmark areas can be relaxed. Some errors in control and sensing are thus permitted in each landmark area without affecting the guaranteedness of the generated motion plan. However, the maximal errors allowed depend on the plan itself and are not known prior to planning.

Another interesting aspect of the method is that, whether it returns success or failure, it always constructs a plan in the form of a non-ordered collection of *reaction rules* described as motion commands associated with regions of the workspace from which the goal can be reliably achieved. This is important in two ways. First, if the input problem has no solution, the robot may nevertheless try to enter one of the regions where a rule is available by performing an initial random motion. Second, if an unexpected event occurs at execution time, the robot may attempt to reconnect to the plan in the same way. The insertion of random motions is an attractive idea when the mean duration of a random motion before it enters one of the regions where reaction rules are available is small enough, i.e. when the total area of these regions is large relative to the workspace area.

In Section 2 (RELATED WORK) we describe how our work relates to previous research. In Section 3 (PLANNING PROBLEM) we precisely state the class of problems solved by our planner and we illustrate this statement with an example. In Section 4 (DIRECTIONAL PREIMAGE BACKCHAINING) we present a first-cut planning method based on the concept of directional preimage backchaining. This method does not provide, however, an efficient way to select the directions of motion for computing preimages. In Section 5 (NON-DIRECTIONAL PREIMAGE BACKCHAINING) we address that issue and we present the actual planning algorithm. In Section 6 (ROBOT NAVIGATION) we discuss various ways for a robot to navigate using a plan generated by our planner. In Section 7 (EXPERIMENTAL RESULTS) we show a series of examples run with the implemented planner. For simplification, Sections 4 through 7 assume that there are no obstacles in the robot's workspace. In Section 8 (DEALING WITH OBSTACLES) we extend the planning method to deal with obstacles, and we present additional experimental results. In Section 9 (DISCUSSION AND EXTENSIONS) we discuss our assumptions and we describe non-implemented extensions of the planner aimed at eliminating the most restrictive ones.

2. Related Work

Our planning method is an instance of the LMT preimage backchaining approach introduced in [26, 28, 15]. The original LMT was targeted toward fine-motion planning for mechanical assembly tasks (part mating operations), but its concepts are more general. In its current stage, our work is more concerned with mobile-robot navigation in two dimensions. But we think that its underlying concepts could also be applied to assembly planning.

The complexity of the general problem addressed by the LMT approach was shown to be nondeterministic exponential-time hard (NEXPTIME-hard) in three dimensions [5], which strongly suggests that planning can take double exponential time in some measure of the size of the problem. To our best knowledge, no lower-bound time complexity result has been established for the two-dimensional problem, but there are several upper-bound results applying to this case. A rather general planning procedure based on algebraic decision techniques is described in [7], which takes double exponential time in the number of steps of the motion plan (in the worst case, this number is itself polynomial in the complexity of the workspace). A less general algorithm obtained by restricting sensory feeback is given in [12], which is simply exponential in the number of steps. A perhaps more practical algorithm is presented in [23], but it is incomplete and nevertheless exponential in the number of steps.

Part of the complexity of LMT, and, more generally, of the motion planning problem in the presence of uncertainty, comes from the subtle interaction between goal reachability and goal recognizability. We not only want the robot to reach the goal despite uncertainty in control; we also want it to recognize goal achievement despite uncertainty in sensing. It is suggested in [15] to simplify planning by assuming partial independence between these two notions. This consists of extracting a subset of the goal that can be unambiguously recognized by the sensors independently of the way it has been achieved. This notion is central to the methods described in [12, 23], as well as to the algorithm presented in this paper. It is also related to the notion of a landmark used in different papers (e.g., [25]). A landmark is a recognizable feature of the workspace that induces a field of influence (if the robot is in this field, it senses the landmark). Various similar notions have also been introduced with different names, e.g. "atomic region" [4], "signature neighborhood" [27], and "perceptual equivalent class" [9, 13]. See also [21].

Our planning algorithm backchains non-directional preimages of the goal. The notion of a non-directional preimage was already present in the original LMT. However, its exact computation was first described in [12, 3], where it was applied without backchaining in order to generate one-step motion strategies. Although our algorithm applies to a different setting (for instance, we allow no compliant motions

sliding along obstacles), it reuses several of the ideas introduced in [12, 3], namely the fact that when the commanded direction of motion varies continuously, the preimage of a goal remains topologically the same, except at some critical directions. These ideas are combined here with the hypothesis that every landmark allows perfect control and sensing within its field of influence, which facilitates backchaining. This combination is the basis of our polynomial-time planning algorithm. A different instantiation of LMT into a polynomial-time planner able to generate multi-step motion plans with control uncertainty is described in [19]. This planner assumes perfect sensing in the boundary of the polygonal workspace, and no sensing elsewhere.

LMT assumes bounded errors and produces guaranteed plans, that is, plans whose success is guaranteed as long as the actual errors during execution stay within these bounds. The concept of a weakly guaranteed plan, which may fail recognizably, is explored in [11]. The concept of a probabilistically guaranteed plan, whose probability of success converges toward one when time grows to infinity, is developed in [16, 18]. The idea of executing a random motion when the planning problem has no guaranteed solution or when the robot encounters unexpected events at execution time is related to this previous work. The notion of a plan described as an unordered collection of reaction rules in the form of motion commands distributed over several regions of the workspace is reminiscent of the concept of a "reaction plan" as proposed in AI planning research [30, 8, 14]. A reaction plan is also represented as a set of control rules allowing an agent to face contingencies at execution time.

Substantial work has been devoted to developing methods computing an optimized estimate of the robot's position while it is moving. For example, techniques have been proposed to combine the estimates provided by both dead-reckoning and environment sensing (e.g., see [2, 10, 24]). However, these techniques address the problem of tracking a selected motion plan as well as possible, not the problem of generating this plan. The goal of planning in the presence of uncertainty is to make sure that executing the plan will reveal enough information to guarantee reliable execution.

3. Planning Problem

The robot is a point moving in a plane, called the *workspace*, containing stationary forbidden circular regions called the *obstacle disks*. The robot can move in either one of two control modes, the *perfect* and the *imperfect* modes.

The perfect control mode can only be used in some stationary circular areas of the workspace called the *landmark disks* (the fields of influence of the landmarks). These disks have null intersection with the obstacle disks. When the robot is in a landmark disk, it knows its position exactly and it has perfect control over its

motions. Some disks may intersect, creating larger areas, called *landmark areas*, through which the robot can move in the perfect control mode. A motion command in the perfect control mode, called a *P-command*, is described by a sequence of via points such that all the *via points* are in the same landmark area, any two consecutive via points are in the same landmark disk, and any two non-consecutive via points are in different landmark disks. The robot can start executing the command only when it is in the landmark disk containing the first via point. It executes the command by moving through the successive via points and stops when it reaches the last one.

A motion command in the imperfect control mode, called an *I-command* is described by a pair $(d, \mathcal{L})$, where $d \in S^1$ is a direction in the plane, called the *commanded direction of motion*, and $\mathcal{L}$ is a set of landmark disks, called the *termination set* of the command. This command can be executed from anywhere in the plane outside the obstacle disks. The robot follows a path whose tangent at any point makes an angle with the direction d that is no greater than some prespecified angle θ called the *directional uncertainty*. The cone of angle 2θ whose axis points along d is called the *directional uncertainty cone*. The robot stops as soon as it enters a landmark disk in $\mathcal{L}$.

The robot has no sense of time, which means that the modulus of its velocity is irrelevant to the planning problem.

The initial position of the robot is known to be anywhere in a specified region $\mathcal{I}$, called the *initial region*, that consists of one or several disks, called the *initial-region disks*. At planning time, we only know that the robot will be in the initial region when the execution of the plan starts; but the robot may not be there yet. Furthermore, we do not want to make any assumption about how it will move into the initial region; perhaps it will be transported there, or it will use another control mode not considered in this paper. Thus, each initial-region disk may be disjoint from the landmark areas, or it may overlap some of them, or it may be entirely contained in one of them. The robot must move into a given region $\mathcal{G}_0$, called the *goal region*, which is any subset, connected or not, of the workspace whose intersection with the landmark disks is easily computable. The problem is to generate a *motion plan*, i.e. an algorithm made up of I- and P-commands, whose execution guarantees that the robot will be in $\mathcal{G}_0$ when the execution of the plan terminates. The robot is not allowed to collide with any of the obstacle disks.

Throughout this paper we let ℓ denote the number of landmark disks scattered across the workspace. We assume that the number of obstacle disks is in $O(\ell)$ and that the number of initial-region disks is small enough to be considered constant. We precompute the set of all landmark areas. Computing a landmark area includes identifying its landmark disks, constructing its boundary as one or several lists of circular arcs, and identifying which initial-region disks are fully contained in the area. The number of circular arcs bounding the union of ℓ disks is linear in ℓ [20].

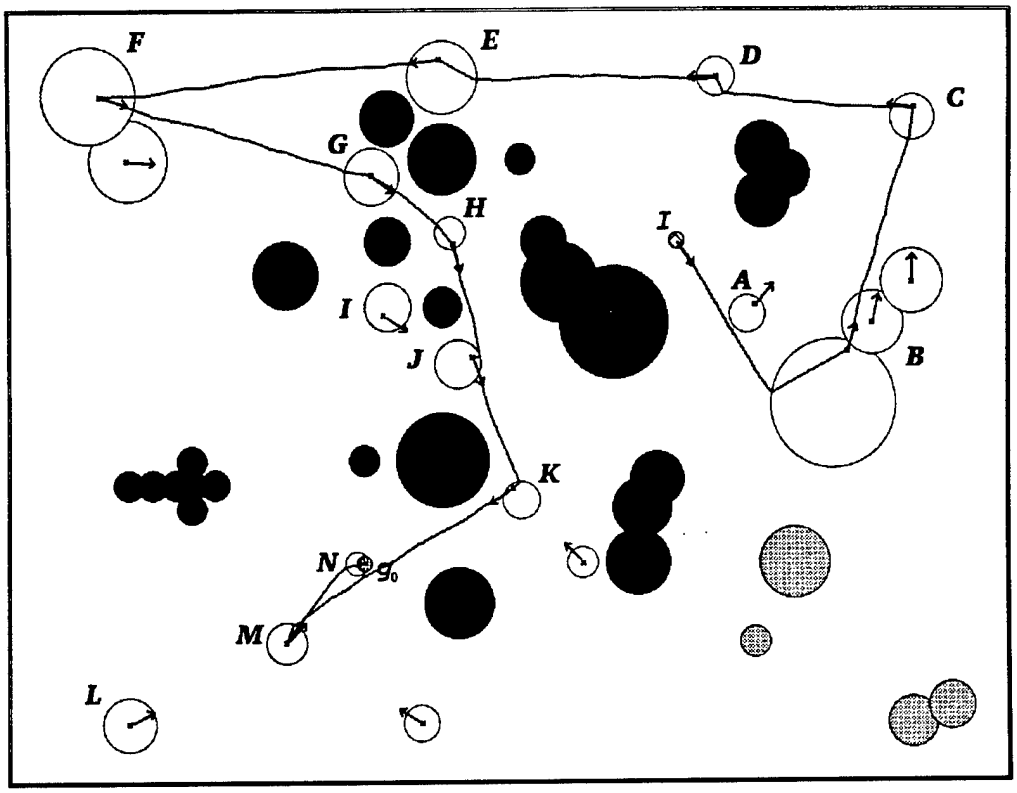


Figure 1: Example of a planning problem

Hence, the total size of the landmark areas' boundaries is $O(\ell)$. The precomputation is carried out using a divide-and-conquer algorithm that takes $O(\ell \log^2 \ell)$ time [29]. We also precompute the intersection of $\mathcal{G}_0$ with the landmark areas. We select a point, called a *goal point*, in the intersection of every landmark area with $\mathcal{G}_0$, if this intersection is not empty. We assume that $\mathcal{G}_0$ is simple enough so that this precomputation takes $O(\ell)$ time. (If the intersection of $\mathcal{G}_0$ with the landmark areas is empty, the goal cannot be achieved reliably, unless it already contains $\mathcal{I}$!)

The planner described in this paper is complete, i.e. it generates a motion plan whenever one exists and returns that no such plan exists otherwise. It takes time $O(s\ell^3 \log \ell)$ and space $O(\ell^3)$, where $s \in O(\ell)$ is the number of landmark areas.

Example: Fig. 1 illustrates the previous description with an example run using the implemented planner. The workspace contains 23 landmark disks (shown white or grey) forming 19 landmark areas, and 25 obstacle disks. The directional uncertainty θ is set to 0.09 radian. The initial and goal regions are two small disks designated

by $\mathcal{I}$ and $\mathcal{G}_0$, respectively.

The white landmark disks are those with which the planner has associated motion commands. The arrow attached to a white disk is the commanded direction of motion of an I-command planned to attain another set of disks. There is at least one arrow per landmark area not intersecting the goal.

The execution of the plan begins with performing the I-command attached to the initial region. When the robot reaches a disk in the termination set of this command, it is guaranteed that a P-command is attached to this disk (hence, it is a white disk in the figure). Executing that P-command allows the robot to attain a point in the current landmark area that is either a goal point (if the goal region intersects this landmark area) or such that an I-command is associated with it (the arrows shown in the figure are drawn from such points). In the first case, plan execution terminates when the goal point is attained. In the second case, the I-command is executed, and so on.

The figure also shows the path produced by a sample execution of the plan. This path first takes the robot from the initial region to the landmark area designated by B. From there, it successively attains and traverses the landmark areas marked C, D, E, F, G, H, J, K, M, and N. The P-command associated with N takes the robot to $\mathcal{G}_0$ where it stops.

The path shown in the figure was produced by the execution of 11 I-commands and 11 P-commands. However, the generated plan could have required up to 12 I-commands. Indeed, the I-command from K is only guaranteed to attain the union of the landmark areas L, M, and N, which form the termination set of the command. Another execution (with different control errors) could have caused the robot to reach L rather than M. The motion command attached to L would then have allowed the robot to reach M. ■

The above problem is a simplification of a real mobile-robot navigation problem, but it is not oversimplified. In Section 9 (DISCUSSION AND EXTENSIONS) we will see that the most restricting assumptions (e.g., that control and sensing are perfect in landmark areas; that landmark and obstacle areas are unions of disks) can be eliminated or made looser, so that the methods described in this paper provide a solid foundation for real mobile-robot navigation.

4. DIRECTIONAL PREIMAGE BACKCHAINING

In this section and the next three we simplify our presentation by assuming that the workspace contains no obstacle disks. Obstacles will be introduced in Section 8 (DEALING WITH OBSTACLES).

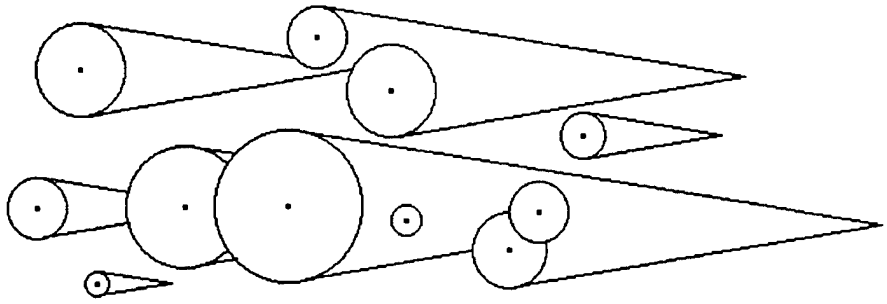


Figure 2: A directional preimage

4.1. Directional Preimage of a Goal

Definition Consider a goal region $\mathcal{G}$. We define the *kernel* of $\mathcal{G}$ as the largest set of landmark disks such that, if the robot is in one of them, it can attain the goal by executing a single P-command. Thus, the kernel of $\mathcal{G}$, denoted by $K(\mathcal{G})$, is the set of all the landmark areas having a non-zero intersection with $\mathcal{G}$. The disks in $K(\mathcal{G})$ are called the *kernel disks*. The other landmark disks are called the *non-kernel disks*.

The *directional preimage* of $\mathcal{G}$, for any given commanded direction of motion d, is the region $P(\mathcal{G}, d)$ defined as the largest subset of the workspace such that, if the robot executes the I-command $(d, K(\mathcal{G}))$ from any position in $P(\mathcal{G}, d)$, then it is guaranteed to reach $K(\mathcal{G})$ and thus to stop in $K(\mathcal{G})$. From the entry point in the kernel, the robot can attain $\mathcal{G}$ by executing a P-command. (Note that $K(\mathcal{G}) \subset P(\mathcal{G}, d)$. If the robot is already in $K(\mathcal{G})$ the I-command $(d, K(\mathcal{G}))$ immediately terminates.)

There is no larger region than $P(\mathcal{G}, d)$ from where the robot is guaranteed to attain $\mathcal{G}$ recognizably by executing one I-command along d followed by a P-command. Indeed, if the robot executes the I-command $(d, K(\mathcal{G}))$ from any position outside this region, it is not guaranteed to reach $K(\mathcal{G})$. From some positions, it may be guaranteed to reach $\mathcal{G}$, but since it may not enter any landmark disk intersecting $\mathcal{G}$, it may not recognize goal achievement and it may thus traverse the goal without stopping.

Names, Labels and Description The directional preimage of a goal $\mathcal{G}$ for any direction $d \in S^1$ consists of one or several connected subsets. Fig. 2 shows an example of a directional preimage with four connected subsets.

Each connected subset in $P(\mathcal{G}, d)$ has no hole, even when the union of the kernel disks has some. Its boundary consists of circular segments called *arcs* and straight

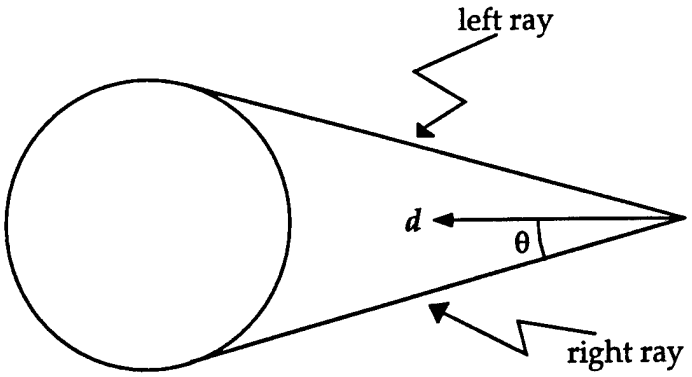


Figure 3: Right and left rays of a landmark disk

segments called *edges*. Each arc is a subset of the boundary of a kernel disk. Let the *right ray* (resp. *left ray*) of a kernel disk L be the half-line tangent to L erected from the tangency point in the direction pointed by $\pi + d + \theta$ (resp. $\pi + d - \theta$), as shown in Fig. 3. Each edge is contained in the right or left ray of some kernel disk, and is called a *right* or *left edge*, accordingly. One extremity of the edge, called its *origin*, is the tangency point of the ray. The other extremity, called the edge's *endpoint*, is the first intersection point of the ray with another kernel disk or another erected ray. The right (or left) ray of any kernel disk thus supports at most one edge of the total preimage's boundary. If two edges share the same endpoint, this endpoint is called a *spike*.

We assign a distinct integer in $[1, \ell]$ to every landmark disk. Using these numbers we give a distinct *name* to every disk, ray and intersection of two rays:
- $\mathtt{d}_i$ is the name of the disk whose number is i,
- $\mathtt{r}_i$ is the name of the right ray of $\mathtt{d}_i$,
- $\mathtt{l}_i$ is the name of the left ray of $\mathtt{d}_i$,
- $\mathtt{x}_{i,j}$ is the name of the intersection of $\mathtt{r}_i$ and $\mathtt{l}_j$.

We *label* every edge in the directional preimage's boundary by the name of the ray supporting it, every arc by the name of the disk it belongs to, and every spike by the name of the corresponding intersection. (Two distinct arcs may receive the same label.)

Except for isolated singular commanded directions of motion where an edge is tangent to a kernel disk or a spike is in contact with a kernel disk (see Fig. 4), every connected subset of a directional preimage is bounded by a simple curve (Jordan curve).

Let the commanded direction of motion be non-singular. Consider a connected subset S of the preimage for this direction. We describe S as the circular list D of

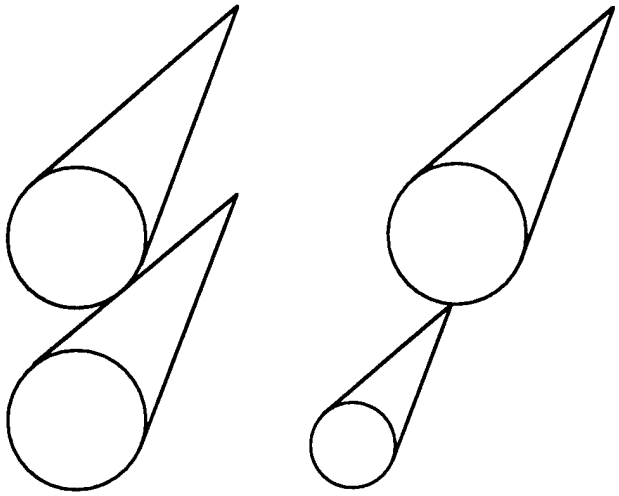


Figure 4: Singular directions

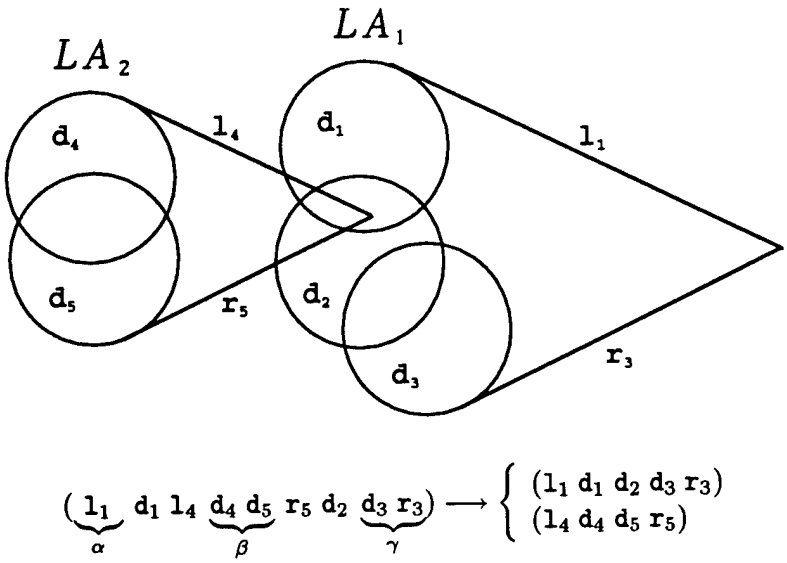


Figure 5: Hidden-spike transformation

the labels met along its boundary as it is traced counterclockwisely. If D is of the form $(\alpha, \mathtt{d}_i, \mathtt{l}_j, \beta, \mathtt{r}_k, \mathtt{d}_l, \gamma)$, such that:
- α, β, and γ are non-empty sublists,

- the endpoints of $\mathtt{l}_j$ and $\mathtt{r}_k$ are in the same landmark area LA, and
- β does not contain the name of any disk in LA,

we say that the intersection of the two rays $\mathtt{l}_j$ and $\mathtt{r}_k$ forms a *hidden spike*. In this case, we apply the following transformation, which is illustrated in Fig. 5: We break D into two lists D_1 and D_2, with $D_1 = (\alpha, \beta', \gamma)$ and $D_2 = (\mathtt{l}_j, \beta, \mathtt{r}_k)$, where β' is the sequence of arc labels encountered while following the boundary of LA counterclockwisely between the last arc of α and the first arc of γ. For every connected subset of the directional preimage, the transformation is repeated until it is applicable to none of the generated lists. The set of lists then obtained is called the *description* of the directional preimage. Each list describes the boundary of a subarea of the preimage, called a *component*. Every component contains a single spike, hidden or not.

At singular directions, we divide the connected subsets with non-simple boundaries into smaller subsets with simple boundaries that touch each other at isolated points, and we then proceed as above.

The kernel $K(\mathcal{G})$ contains $O(\ell)$ disks. Its boundary contains $O(\ell)$ arcs. For any $d \in S^1$, the boundary of $P(\mathcal{G}, d)$ contains $O(\ell)$ edges. Indeed, each ray of a kernel disk supports at most one edge. Therefore, the size of the description of $P(\mathcal{G}, d)$ is $O(\ell)$. The number of spikes is $O(s)$, where $s \in O(\ell)$ is the number of landmark areas. Thus:

Lemma 1 *The description of a directional preimage has size $O(\ell)$.*

Computation We erect the $O(\ell)$ rays tangent to the precomputed boundary of $K(\mathcal{G})$ and pointing along the directions $\pi + d \pm \theta$. We compute the boundary of every connected subset of $P(\mathcal{G}, d)$, in the form of a list of disk and ray names, by sweeping a line perpendicular to d, in the direction of $d + \pi$ [29, 22]. Each ray is interrupted where it first intersects a kernel disk or another ray. This computation takes $O(\ell \log \ell)$ time.

Every landmark area in $K(\mathcal{G})$ is contained in one and only one connected subset of the preimage. During the sweep we keep track of the edge endpoints in every landmark area. When the sweep is completed, we sort the endpoints in every landmark area in counterclockwise order in a cyclic list. Whenever the endpoint of a left edge immediately precedes the endpoint of a right edge in such a list, the rays supporting these two edges form a hidden spike and we apply the transformation described in the previous paragraph. The combined cost of all the transformations is $O(\ell \log \ell)$. Hence:

Lemma 2 *The description of a directional preimage is computed in time $O(\ell \log \ell)$.*

4.2. First-Cut Planning Algorithm

Backchaining Assume that we select d_0 such that the directional preimage of the problem's goal $\mathcal{G}_0$ contains the initial region $\mathcal{I}$. We then have a motion plan to achieve the goal. Indeed, from its initial position in $\mathcal{I}$, the robot can attain the kernel $K(\mathcal{G}_0)$ by executing the I-command $(d_0, K(\mathcal{G}_0))$. Then, by switching to the perfect control mode, it can reach the goal without leaving $K(\mathcal{G}_0)$.

However, in general, such a "one-step" motion plan does not exist. If $K(\mathcal{G}_0)$ is empty, so is $P(\mathcal{G}_0, d)$ for any $d \in S^1$; then the planner can safely return failure (if $\mathcal{I} \not\subset \mathcal{G}_0$). If $K(\mathcal{G}_0)$ is not empty and $P(\mathcal{G}_0, d_0)$, for the selected direction d_0, does not contain $\mathcal{I}$, we can treat $P(\mathcal{G}_0, d_0)$ as an intermediate goal $\mathcal{G}_1$ and try to produce a motion plan to achieve it from $\mathcal{I}$. This means that we compute the kernel $K(\mathcal{G}_1)$ and, if it is a proper superset of $K(\mathcal{G}_0)$, the preimage $P(\mathcal{G}_1, d_1)$ for some direction $d_1 \in S^1$. If we select d_1 such that $P(\mathcal{G}_1, d_1)$ contains $\mathcal{I}$, we then have a two-step motion plan to achieve $\mathcal{G}_0$; otherwise, we can consider $P(\mathcal{G}_1, d_1)$ as a new intermediate goal $\mathcal{G}_2$, and so on. The whole process is called *directional preimage backchaining.*

Planning P-commands Let LA be a landmark area in $K(\mathcal{G}_0)$ and G be the goal point in LA. We construct a tree, called the *P-command tree* of LA, whose nodes are all the disks in LA. The root of the tree is the disk containing G and any two disks related by a link of the tree overlap. (Any tree verifying these properties is adequate.) We select a via point in the intersection of every disk other than the root with its immediate parent in the tree. The P-command tree of LA will be used at execution time to select the P-command to execute when the robot enters a disk in LA. The P-command will simply be the sequence of via points collected by tracing the path in the tree between the entered disk and the root, with the goal point G added at the end of the sequence. A P-command tree is constructed for every landmark area in $K(\mathcal{G}_0)$.

Consider now the kernel $K(\mathcal{G}_i)$ of an intermediate goal $\mathcal{G}_i$ $(i > 0)$. In every landmark area $LA \subseteq K(\mathcal{G}_i) \backslash K(\mathcal{G}_{i-1})$, we pick a disk that has a non-zero intersection with $\mathcal{G}_i$ and a point in this intersection. This point is called the *exit point* of LA. In the same way as above, we construct the P-command tree of LA, with the disk containing the exit point as the root. At execution time, if the robot attains this exit point, the generated plan prescribes to immediately switch to executing the I-command $(d_{i-1}, K(\mathcal{G}_{i-1}))$.

A straightforward computation of all the P-command trees takes $O(\ell^2)$ time in total.

Line Sweep Directional preimage backchaining requires every newly computed preimage to be checked for containment of the initial region $\mathcal{I}$. If it does not contain $\mathcal{I}$, the new preimage must also be checked for intersection with landmark disks not in the current goal's kernel. These computations can be incorporated in the sweep-line algorithm that constructs the preimage.

The augmented algorithm generates the description of the intersection of the preimage with every initial-region disk I in a data structure attached to I consisting of a boolean value, IN, and two sets, Right-Cut and Left-Cut. The value of IN is `true` if I is entirely contained in the preimage and `false` otherwise. The Right-Cut (resp. Left-Cut) set contains the label of every right (resp. left) edge intersecting I. The initial region is contained in the preimage when the IN value of every initial-region disk is `true`. (The Right-Cut and Left-Cut sets will be used later.)

The description of the intersection of the preimage with the non-kernel disks is simpler. It is generated as the set of all non-kernel disks intersected by the preimage.

During the sweep we remove any non-kernel disk from further consideration, as soon as we detect that it is intersected by an edge of the preimage being computed. With this simplification, the time complexity of the sweep-line algorithm remains $O(\ell \log \ell)$.

Choice of Directions In order to transform directional preimage backchaining into an effective planning algorithm, we still need a method for choosing a direction of motion at every iteration of the backchaining process. One simple method would be to discretize the continuous set S^1 into a finite set of regularly spaced directions and search a graph by trying all possible combinations of directions in this finite set. But the resulting planning algorithm would not be complete in general, even if we used a very fine discretization. In the worst case, it would also require exponential time in the number of landmark areas.

We solve the above issue by using the notion of a non-directional preimage and modifying the preimage backchaining process accordingly, as described in the next section.

5. Non-Directional Preimage Backchaining

5.1. Non-Directional Preimage of a Goal

Definition Let us consider the directional preimage $P(\mathcal{G}, d)$ when d varies continuously over S^1. For every value of d, we are interested in answering the questions "Does $P(\mathcal{G}, d)$ include $\mathcal{I}$?" and "What non-kernel disks does $P(\mathcal{G}, d)$ intersect?". Although there are infinitely many possible values of d, we will see below that these answers change at a finite number of *critical* directions. In order to detect these

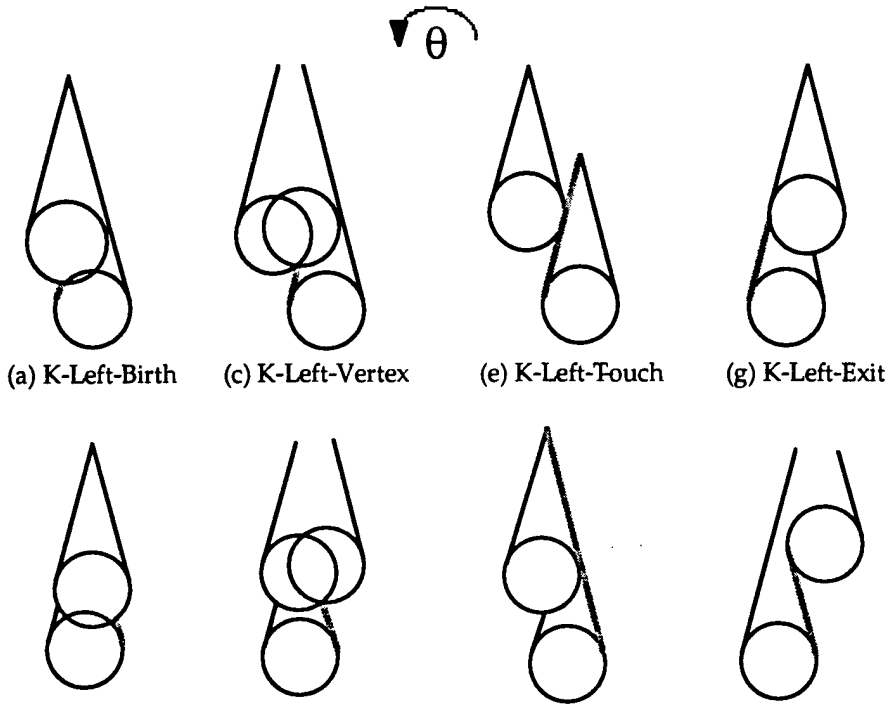


Figure 6: Events responsible for D-critical directions

changes we must also track the variation of $P(\mathcal{G}, d)$. Fortunately, $P(\mathcal{G}, d)$ varies continuously and keeps the same description, except at a finite number of critical directions, many of which are different from the previous ones. Each open angular interval of S^1 between any two consecutive critical directions is called a *regular interval.*

Let $(d_{c_1}, \ldots, d_{c_p})$ be the cyclic list of all critical directions in counterclockwise order and $I_1, \ldots, I_p$ be the regular intervals between them, with $I_i = (d_{c_i}, d_{c_{i+1 (\mathrm{mod}\ p)}})$. For any interval I_i, let d_{nc_i} be any direction in I_i. In order to characterize all the directional preimages of $\mathcal{G}$ and their intersection with $\mathcal{I}$ and the non-kernel disks, it suffices to compute $P(\mathcal{G}, d)$ for all $d \in \{d_{nc_1}, d_{c_1}, d_{nc_2}, \ldots, d_{c_p}\}$. The set $NP(\mathcal{G})$ of all these directional preimages is called the *non-directional preimage* of $\mathcal{G}$ [12].

Every critical direction corresponds to an *event* caused by the motion of an edge or a spike of the current directional preimage, e.g. an edge hits a landmark disk. We describe below all such events. We assume that the landmark and initial-region disks are in general position, i.e. no two events occur simultaneously.

We call the critical directions where the description of the directional preimage does change the *D-critical* directions. The critical directions where the description of the intersection of the directional preimage with the initial-region disks changes, other than the D-critical directions, are called the *I-critical* directions. The critical directions where the intersection with non-kernel disks changes, other than the D-critical directions, are the *L-critical* directions.

D-Critical Directions As d varies counterclockwisely, the description of a directional preimage $P(\mathcal{G}, d)$ changes when and only when one of the following events occur (see Fig. 6):[1]
- A *K-Left-Birth* event occurs when a new left edge emerges at the intersection of two kernel disks.
- A *K-Right-Death* event occurs when a right edge disappears at the intersection of two kernel disks.
- A *K-Left-Vertex* event occurs when the endpoint of a left edge crosses the intersection between two kernel disks.
- A *K-Right-Vertex* event occurs when a right edge crosses the intersection between two kernel disks.
- A *K-Left-Touch* event occurs when a left edge reaches a kernel disk by becoming tangent to it.
- A *K-Right-Touch* event occurs when a right edge reaches a kernel disk by becoming tangent to it.
- A *K-Left-Exit* event occurs when a left edge leaves the kernel disk containing its endpoint by becoming tangent to it.
- A *K-Right-Exit* event occurs when a right edge leaves the kernel disk containing its endpoint by becoming tangent to it.

There are $O(\ell)$ rotating rays. The boundary of $K(\mathcal{G})$ has $O(\ell)$ arcs. Hence, there are $O(\ell)$ K-Left-Birth and K-Right-Death events, and $O(\ell^2)$ K-Left-Vertex and K-Right-Vertex events. There also are $O(\ell^2)$ K-Left-Touch, K-Right-Touch, K-Left-Exit and K-Right-Exit events. Therefore:

Lemma 3 *There are $O(\ell^2)$ D-critical directions.*

I-Critical Directions As d rotates counterclockwisely, the description of the intersection of $P(\mathcal{G}, d)$ with initial-region disks may change at some D-critical values of d. It also changes at I-critical directions corresponding to the following events

[1] All these events are due to kernel disks and, for this reason, we call them K-... events. In Section 8 we will define other D-critical directions, due to obstacle disks; we will call the corresponding events O-... events.

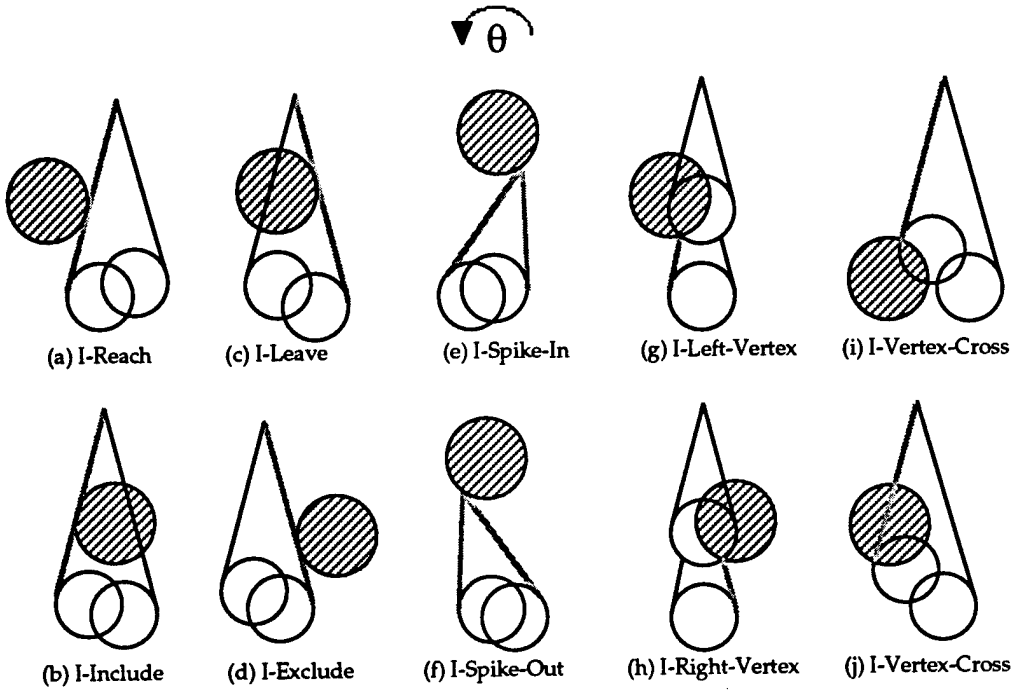


Figure 7: Events responsible for I-critical directions

(see Fig. 7, where the striped disk is an initial-region disk):
- An *I-Reach* event occurs when a left edge reaches an initial-region disk by becoming tangent to it.
- An *I-Include* event occurs when a left edge leaves an initial-region disk by becoming tangent to it.
- An *I-Leave* event occurs when a right edge reaches an initial-region disk by becoming tangent to it.
- An *I-Exclude* event occurs when a right edge leaves an initial-region disk by becoming tangent to it.
- An *I-Spike-In* event occurs when a spike enters an initial-region disk.
- An *I-Spike-Out* event occurs when a spike exits an initial-region disk.
- An *I-Left-Vertex* event occurs when a left edge crosses the intersection of the kernel disk containing its endpoint with an initial-region disk.
- An *I-Right-Vertex* event occurs when a right edge crosses the intersection of the kernel disk containing its endpoint with an initial-region disk.
- An *I-Vertex-Cross* event occurs when the origin of an edge crosses the intersection of its kernel disk with an initial-region disk.

The number of initial-region disks is assumed constant. Hence, events other than I-Spike-In and I-Spike-Out produce $O(\ell)$ I-critical directions. I-Spike-In and I-Spike-Out events create $O(\ell^2)$ I-critical directions. Hence:

Lemma 4 *There are $O(\ell^2)$ I-critical directions.*

L-Critical Directions As d rotates counterclockwisely, the description of the intersection of $P(\mathcal{G}, d)$ with non-kernel disks may change at some D-critical values of d. It may also change at L-critical directions corresponding to the following events:
- An *L-Reach* event occurs when a left edge reaches a non-kernel disk by becoming tangent to it.
- An *L-Include* event occurs when a left edge leaves a non-kernel disk by becoming tangent to it.
- An *L-Leave* event occurs when a right edge reaches a non-kernel disk by becoming tangent to it.
- An *L-Exclude* event occurs when a right edge leaves a non-kernel disk by becoming tangent to it.
- An *L-Spike-In* event occurs when a spike enters a non-kernel disk.
- An *L-Spike-Out* event occurs when a spike exits a non-kernel disk.

There are $O(\ell^2)$ L-Reach, L-Include, L-Leave, L-Exclude events and $O(\ell^3)$ L-Spike-In and L-Spike-Out events. Hence:

Lemma 5 *There are $O(\ell^3)$ L-critical directions.*

Since we are only interested in knowing that a non-kernel disk is intersected by some directional preimage and in computing a direction for which this happens, we will consider only the L-Reach and L-Spike-In events in the following.

Computation of Non-Directional Preimage See Subsection 5.3.

5.2. Planning Method

Plan Generation If $\mathcal{I} \not\subset \mathcal{G}_0$, the planner first computes the kernel $K(\mathcal{G}_0)$. If $K(\mathcal{G}_0)$ is empty, the planner returns failure. Otherwise, it associates a P-command to reach a goal point with every landmark disk in this kernel. If $\mathcal{I} \subset K(\mathcal{G}_0)$ the planner returns success.

Let us assume that $\mathcal{I} \not\subset K(\mathcal{G}_0)$. The planner then computes the non-directional preimage $NP(\mathcal{G}_0)$. If $NP(\mathcal{G}_0)$ contains a directional preimage $P(\mathcal{G}_0, d)$ that includes $\mathcal{I}$, then the planner attaches the motion command $(d, K(\mathcal{G}_0))$ to $\mathcal{I}$ and returns

success. Otherwise, for every landmark area $LA \not\subset K(\mathcal{G}_0)$ that has a non-zero intersection with a directional preimage $P(\mathcal{G}_0, d)$ in $NP(\mathcal{G}_0)$, an exit point is arbitrarily selected in $LA \cap P(\mathcal{G}_0, d)$ and the I-command $(d, K(\mathcal{G}_0))$ is attached to this point. (If the same area LA intersects several directional preimages, only one intersection is used to produce the I-command.) The union of the directional preimages in $NP(\mathcal{G}_0)$ is now considered as an intermediate goal $\mathcal{G}_1$.

The kernel $K(\mathcal{G}_1)$ is constructed. By construction, $K(\mathcal{G}_1) \supseteq K(\mathcal{G}_0)$. If $K(\mathcal{G}_1) = K(\mathcal{G}_0)$, the planner terminates with failure since it cannot compute a larger non-directional preimage than $NP(\mathcal{G}_0)$. Otherwise, every landmark area in $K(\mathcal{G}_1) \backslash K(\mathcal{G}_0)$ contains one disk L with an exit point and a motion command attached to it. With every other disk in the landmark area, the planner associates a P-command to reach the exit point in L. If $\mathcal{I} \subset K(\mathcal{G}_1)$ the planner returns success, else it computes the non-directional preimage of $\mathcal{G}_1$, and so on.

During this backchaining process, the set of landmark areas in the kernels of the successive goals increases monotonically. At every iteration, either there is a new landmark area in the kernel, and the planner proceeds further, or there is no new area, and the planner terminates with failure. The planner terminates with success whenever it has constructed a kernel $K(\mathcal{G}_n)$ containing $\mathcal{I}$ or a non-directional preimage $NP(\mathcal{G}_n)$ that includes a directional preimage containing $\mathcal{I}$. Let s be the number of landmark areas. The number of iterations is bounded by s. Thus, $n \leq s$.

Plan Execution Assume that the planner returns success after computing a non-directional preimage $NP(\mathcal{G}_n)$ that contains $\mathcal{I}$. The generated plan can be regarded as a non-ordered collection of *reaction rules*. Each rule is a motion command whose execution is conditional to the entry of the robot into a region of the workspace, either the initial region, or a landmark disk, or an exit point:
- The rule associated with $\mathcal{I}$ is the I-command $(d_n, K(\mathcal{G}_n))$, where d_n is such that $\mathcal{I} \subset P(\mathcal{G}_n, d_n)$.
- The rule associated with a landmark disk is a P-command to attain the exit point or the goal point of the landmark area to which the disk belongs.
- The rule attached to the exit point of each landmark area LA contained in $K(\mathcal{G}_{i+1}) \backslash K(\mathcal{G}_i)$, for any $i \in [0, n-1]$, is an I-commmand $(d_i, K(\mathcal{G}_i))$, where d_i is such that $LA \cap P(\mathcal{G}_i, d_i)$ is a non-empty region containing the exit point.

The plan is executed as follows: The robot first executes the I-command associated with the initial region. This command guarantees that the robot will stop in a landmark disk with a P-command attached to it. Then the robot executes this P-command, and attains a goal point or an exit point. If it attains a goal point, the execution of the plan is terminated. If it attains an exit point, the I-command attached to this point is executed. This command leads the robot to a new landmark disk, and so on.

An exit point was selected by the planner in a landmark area when this area intersected a directional preimage for the first time. Later, the planner never changed the command attached to this point. Therefore, when the robot executes the I-command $(d_i, K(\mathcal{G}_i))$ from the exit point of some landmark area LA, no landmark disk in LA can possibly be in the termination set $K(\mathcal{G}_i)$ of this command. Thus, since $K(\mathcal{G}_0) \subset K(\mathcal{G}_1) \subset \ldots \subset K(\mathcal{G}_n)$, the robot cannot terminate its motion in the same landmark area twice by executing the plan. Hence, it is guaranteed to reach $\mathcal{G}_0$ after executing an alternate sequence of I- and P-commands whose length is smaller than or equal to $2(n+1)$.

If the planner returns success when $K(\mathcal{G}_n)$ contains $\mathcal{I}$, the generated plan is essentially the same as above, except that it does not include a rule associated with $\mathcal{I}$, since the robot will already be in a landmark area having a P-command attached to it.

Completeness and Optimality The execution of any motion plan causes a sequence of I- and P-commands to be executed. Since one can always merge any two consecutive P-commands into a single equivalent one, we say that this sequence is k-step if it contains exactly k I-commands.

By definition of the kernel of any goal $\mathcal{G}_i$, $K(\mathcal{G}_i)$ is the maximal subset of the workspace from which the robot can reliably achieve $\mathcal{G}_i$ in zero steps. By definition of the non-directional preimage of $\mathcal{G}_i$, the set $NP(\mathcal{G}_i) \cup K(NP(\mathcal{G}_i))$, i.e. the union of the non-directional preimage of $\mathcal{G}_i$ and its kernel, is the maximal subset from which the robot can reliably achieve $\mathcal{G}_i$ in at most one step.

Hence, $K(\mathcal{G}_0)$ is the maximal subset from where the robot can reliably achieve the goal of the problem in zero steps, and $NP(\mathcal{G}_i) \cup K(NP(\mathcal{G}_i))$, for any $i \geq 0$, is the maximal subset from where the robot can reliably achieve $\mathcal{G}_0$ in at most $i+1$ steps. Thus, if the goal $\mathcal{G}_0$ can reliably be achieved from $\mathcal{I}$, iterative backchaining of non-directional preimages from $\mathcal{G}_0$ is guaranteed to terminate with success. If $\mathcal{G}_0$ cannot reliably be achieved from $\mathcal{I}$, backchaining will terminate with failure, since the number of iterations is bounded by the number of landmark areas. Thus, the planner is complete.

Let a motion plan be *optimal* if the maximal number of steps required by its execution is minimal over all possible motion plans that are guaranteed to reliably achieve the goal $\mathcal{G}_0$. The maximal number of steps for a plan produced by our planning algorithm is equal to the number of backchaining iterations before a kernel or a preimage contains the initial region. By definition of the non-directional preimages, the number of iterations is equal to the minimal number of steps that is required to achieve the goal in the worst case. Hence, our algorithm generates optimal plans. Furthermore, after the execution of any sequence of steps, the subset of the motion plan that may still be used to attain the problem's goal is also optimal.

Theorem 1 *The planning algorithm is complete and generates optimal plans.*

5.3. Computation of a Non-Directional Preimage

Overview The above planning method does not require that we compute the full description of the non-directional preimage of the current goal at every iteration. It is sufficient to:
- determine whether there is a direction for which the directional preimage fully contains $\mathcal{I}$, and
- identify all the non-kernel disks that have a non-zero intersection with at least one directional preimage.

If a directional preimage for some non-critical direction contains the initial region (or intersects a non-kernel disk L), the directional preimage for the critical direction just before or just after this non-critical direction contains $\mathcal{I}$ (or intersects L) as well. Hence, it is sufficient to consider the directional preimages at the critical directions.

We first compute the directional preimage of $\mathcal{G}$ for a direction d_α arbitrarily selected in S^1. Using the augmented sweep-line algorithm of Subsection 4.2, we also compute the description of its intersection with initial-region disks and non-kernel disks.

The rest of the computation is an alternation of two phases: *event scheduling* and *event processing*. Event scheduling consists of using the current directional preimage description to schedule potential new events by inserting the corresponding directions in a list OPEN that is sorted in counterclockwise order. Event processing consists of removing the first direction in OPEN, checking that it actually is a critical direction, and updating the description of the directional preimage and its intersection with the initial-region and non-kernel disks. The alternation ends when OPEN is empty.

We detail these two computation phases below. The names and labels given to rays, ray intersections, edges, and spikes (see Subsection 4.1) are used to track them across different values of the commanded direction of motion. For example, two rays, for two different directions, are the same "object" if they have the same name.

Event Scheduling Using the description of the first directional preimage $P(\mathcal{G}, d_\alpha)$, we identify potential critical directions by considering all edges and spikes in this preimage. A direction d_{pc} is potentially critical if the ray supporting an edge of $P(\mathcal{G}, d_\alpha)$ is tangent to a landmark disk or an initial-region disk at direction d_{pc} or passes through the intersection of the boundaries of two disks at d_{pc}, or if the curve traced by the intersection of two rays, when the direction varies, intersects an initial-region or a non-kernel disk at direction d_{pc}. The type of the potential event associated with the direction d_{pc} is uniquely defined in advance. For example:

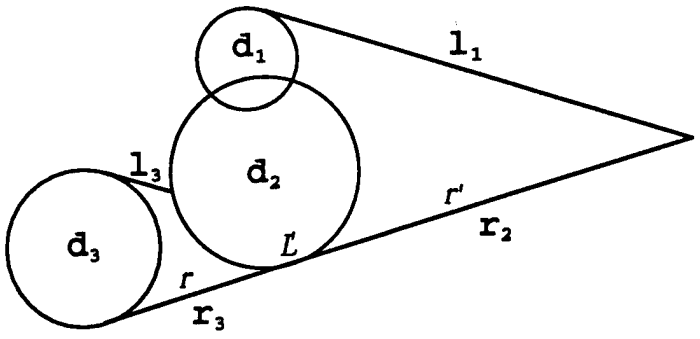


Figure 8: Updating the directional preimage at a K-Right-Touch event

- If the ray l supporting a left edge of $P(\mathcal{G}, d_\alpha)$ is tangent to a kernel disk L at d_{pc}, the type of the event is K-Left-Touch if L is on the left of l and K-Left-Exit if it is on the right.
- If the curve traced by a spike of $P(\mathcal{G}, d_\alpha)$ cuts an initial-region disk at d_{pc}, the type of the event is I-Spike-In if the curve enters the disk and I-Spike-Out if it exits the disk.

Notice, however, that a scheduled event may not occur since the edge or the spike used to schedule it may no longer be part of the current directional preimage at direction d_{pc}.

We mark the label identifying every processed edge and every processed spike. Later on, for every new directional preimage computed at a critical direction, for every edge or spike of this preimage whose label is not marked yet, we perform the same computation as above and mark its label. (Note that all critical directions caused by K-Left-Birth events are scheduled at the beginning, since these directions only depend on the kernel.)

The curve traced by the intersection of a right and a left ray is a circle if the two rays are tangent to the same disk. Otherwise it is a fourth-degree curve. The algebraic expression of the intersection of this curve with a circle is given in the Appendix at the end of this paper.

Event Processing Let d_{pc} be a direction removed from OPEN. The first step is to verify that this direction is actually a critical one. For example, assume that d_{pc} corresponds to a K-Right-Touch event. Hence, the right ray r of a kernel disk is colinear with the right ray r' of another kernel disk L'. In order for the K-Right-Touch event to effectively occur, the ray r must support an edge of the current

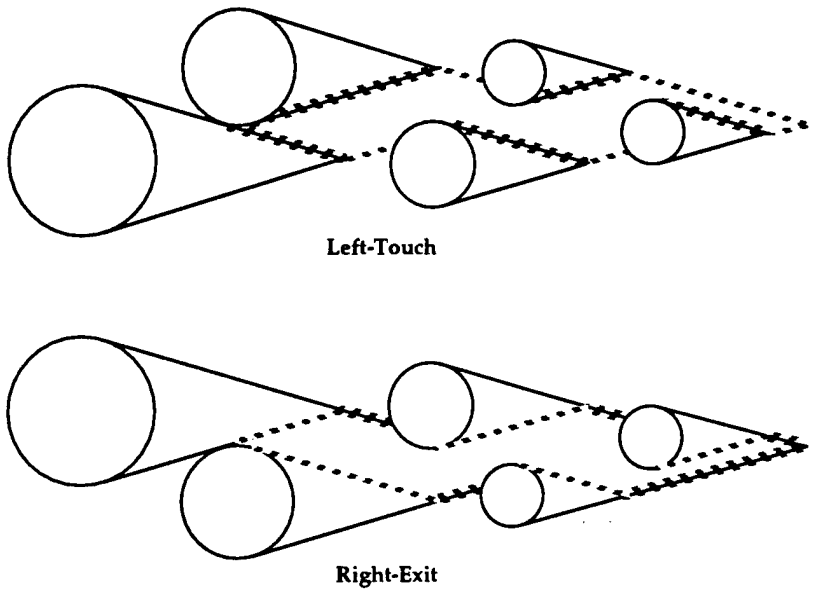


Figure 9: Catastrophic K-Left-Touch and K-Right-Exit events

directional preimage and this edge's length must be greater than or equal to the distance between the origins of r and r'. Checking whether d_{pc} is a critical direction takes constant time for all types of events.

The second step is carried out only if d_{pc} is a D-critical direction. It consists of updating the description of the directional preimage according to the occurring event. For all events, except K-Left-Touch and K-Right-Exit events, the change is small and can be computed in constant time. For example, assume that the above K-Right-Touch event occurs. The description of the directional preimage is then modified by inserting the names of L' and r' in sequence immediately after the name of r (see Fig. 8). On the other hand, a K-Left-Touch or K-Right-Exit event may result in a modification requiring $O(\ell)$ time to compute (see Fig. 9). When this occurs we say that the event is *catastrophic*. We recompute the directional preimage from scratch at every K-Left-Touch or K-Right-Touch event, using the sweep-line algorithm.

The third step is carried out at every D- and I-critical orientation to update the description of the intersection of the directional preimage with $\mathcal{I}$ and check if the new preimage contains $\mathcal{I}$. This consists of updating the data structure (IN,Right-Cut,Left-Cut) attached to every initial-region disk. For example, when an I-Spike-In event occurs involving an initial-region disk I and a spike s, the names of the right

and left rays of s are inserted in the Right-Cut and Left-Cut sets of I, respectively. This update is carried out at every D-critical and I-critical direction. It takes constant time for all event types, except K-Left-Touch or K-Right-Exit. But then the sweep-line algorithm recomputes the full data structure.

The fourth step is carried out at every D- and L-critical orientation to determine which non-kernel disks are intersected by the new directional preimage. Since a non-kernel disk will be included in the kernel of the non-directional preimage being computed if it intersects one of its directional preimages, we remove a non-kernel disk from further consideration as soon as it intersects a directional preimage. Therefore, when we process events, we only have to consider K-Left-Touch, K-Right-Exit, L-Reach, and L-Spike-In events. At K-Left-Touch and K-Right-Exit events, the sweep-line algorithm determines which non-kernel disks are intersected by the new directional preimage. At L-Reach and L-Spike-In events the newly intersected non-kernel disk is identified in constant time.

5.4. Complexity Analysis

At every iteration, the planner computes a directional preimage, schedules events, and processes them.

Computing a directional preimage takes time $O(\ell \log \ell)$, including the time for generating the current goal's kernel.

Scheduling all the events, except the L-Spike-In ones, takes time $O(\ell^2 \log \ell)$. Indeed, there are $O(\ell^2)$ potential events of these types in total, and we must sort the corresponding directions. But there are $O(\ell^3)$ L-Spike-In events and it takes $O(\ell^3 \log \ell)$ time to sort the corresponding critical directions. One may remark that we schedule some events too early. If the scheduling of these events were postponed, the need for scheduling them could disappear in the meantime, thus reducing the number of scheduled events. Some straightforward improvements are possible in this way (and have been incorporated in our implementation), but they do not modify the asymptotic time complexity of event scheduling.

Processing all the potential events takes $O(\ell^3 \log \ell)$ time. The dominant cost is that of processing the K-Left-Touch and K-Right-Exit events, because we recompute the directional preimage at each of them. The cost of processing all the other events is only linear in the number of these events, hence $O(\ell^3)$ for the L-Spike-In events and $O(\ell^2)$ for all others.

Therefore, the time complexity of a planner iteration is $O(\ell^3 \log \ell)$. Since there are at most s iterations, the planner runs in time $O(s\ell^3 \log \ell)$. We can compute all the P-command trees at the end in time $O(\ell^2)$. The overall computation requires space $O(\ell^3)$. The dominant cost here is that of storing the potential critical directions (list OPEN).

Theorem 2 *The planning algorithm takes time* $O(s\ell^3 \log \ell)$ *and space* $O(\ell^3)$.

The time complexity of an iteration of the planner essentially results from the treatment of L-Spike-In, K-Left-Touch and K-Right-Exit events. Concerning L-Spike-In events, we can compute the $O(\ell^3)$ directions where they may occur in advance, so that we sort them only once and reuse this result at every iteration of the planner. Processing the L-Spike-In events then takes $O(s\ell^3)$ time over all $O(s)$ iterations. On the other hand, in a way similar to the one presented in [3], we can show that, at every iteration, although each of the $O(\ell^2)$ K-Left-Touch and K-Right-Exit events may cause $O(\ell)$ changes in the directional preimage, the total number of changes caused by all the K-Left-Touch and K-Right-Exit events is $O(\ell^2)$. We can compute these changes in $O(\ell^2 \log \ell)$ time by using an additional data structure that takes $O(\ell^2 \log \ell)$ time to create and update. Updating the intersection of the directional preimages with the initial-region and non-kernel disks can be achieved within this time complexity. All these modifications combined reduce planning time to $O(s\ell^3 + \ell^3 \log \ell)$, which is a relatively small improvement over the above time complexity.

6. Robot Navigation

6.1. Case Where the Planner Returns Success

See Subsection 5.2, paragraph **Plan Execution.**

6.2. Case Where the Planner Returns Failure

The planner returns failure when the kernel of the current goal, call it $\mathcal{G}_{m+1}$, is equal to the kernel of the previous goal, i.e. $K(\mathcal{G}_{m+1}) = K(\mathcal{G}_m)$. No I-command is then attached to the initial region. But an incomplete plan has nevertheless been generated, in the form of a collection of reaction rules attached to all landmark disks (and exit points) from which it is possible to reliably achieve $\mathcal{G}_0$. The robot may attempt to attain the goal by using this plan.

Let us assume that there exists a third control mode, called the *random control mode.* A motion command in this mode, called an *R-command*, only specifies a termination set $\mathcal{L}$ of landmark disks. When the robot executes this command, it performs a Brownian motion with mean 0, until it enters any disk in $\mathcal{L}$; then it stops.

The probability of a Brownian motion in the plane to enter a disk of non-zero radius converges to 1 when the duration of the motion grows toward infinity. (The hypothesis that the robot has no sense of time does not mean that time does not

exist.) Therefore, the robot could construct $\mathcal{L}$ as the set of all landmark disks which have a reaction rule attached to them and execute the R-command that stops in $\mathcal{L}$. The robot is then guaranteed to attain a disk in $\mathcal{L}$. When this happens, it shifts to executing the incomplete plan generated by the planner. Neither the duration of the execution of the initial R-command, nor even its expected value, are bounded, however.

Let us assume that the robot workspace is bounded by a wall forming a smooth simple curve W enclosing all the landmark disks. Let a motion in the random control mode be a Brownian motion with reflection on W [1]. This means that, if the robot hits the wall, its motion is reflected symmetrically to the tangent of W at the hitting point. Then, in principle, we can bound the expected duration of a Brownian motion starting anywhere in the initial region and terminating as soon as it enters a disk in $\mathcal{L}$. Using this bound, the robot's navigation system may decide whether it is worth to execute an R-command.

A Brownian motion with reflection on W is guaranteed to reach any landmark disk provided that we wait long enough. In particular, it is guaranteed to reach $K(\mathcal{G}_0)$. Therefore, the robot could achieve the goal without planning. The role of planning is to bound the time necessary to achieve the goal (if the planner returns success), or reduce the expected time before entering a landmark disk from which a guaranteed plan to the goal exists (if the planner returns failure). We could also make the planner evaluate the expected time to attain any landmark disk in the kernel of the current goal at every iteration, and stop planning when this time is small enough. This may be useful if the time allocated to planning is limited.

An alternative to executing an uninformed random motion from the initial region is to run the planner with a reduced directional uncertainty θ until it finds a plan from the initial region. This alternative will be discussed in Subsection 9.3.

6.3. Unexpected Events

Let us assume that the hypotheses concerning motion control and landmark sensing stated in the problem description of Section 3 are only almost always correct, so that unexpected events may occur during plan execution. For example, the robot's wheels may slip on the ground yielding a directional error greater than θ; or the robot may accidentally be pushed outside a landmark area while it was executing a P-command; or, a landmark may have been inadvertently "turned off", so that for a while it cannot be sensed by the robot. This event may have no noticeable effect on the execution of the plan, in which case it is harmless to continue executing the plan. The event may also lead the robot to enter a landmark area that it was not expected to enter, or reach the wall W bounding the workspace, or become senseless while it was expected to sense a landmark. The robot then detects that something

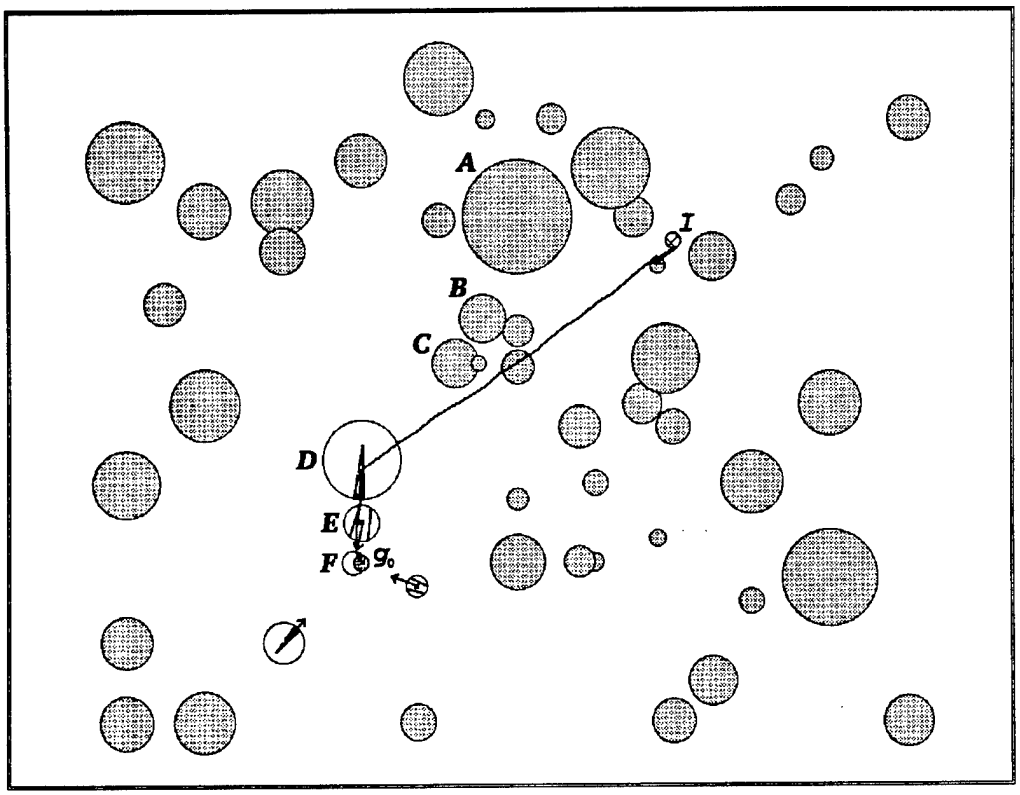


Figure 10: Example with $\theta = 0.1$ radian

abnormal happened. It does not make sense for it to continue the execution of the plan. One way to proceed is to execute an R-command terminating in any landmark disk with a reaction rule attached to it, and resume executing the plan afterward. Executing random motions is a general approach to recover from unexpected events, but it can only be efficient if unexpected events remain exceptional [16].

7. Experimental Results

We implemented the above planning and navigation techniques, along with a robot simulator, in C on a DECstation 5000. The implemented planner incorporates two significant improvements:

(1) During the computation of a non-directional preimage, it does not discard a non-kernel disk L as soon as this disk intersects a directional preimage. Instead, it determines the intervals of all directions at which the directional preimage intersects

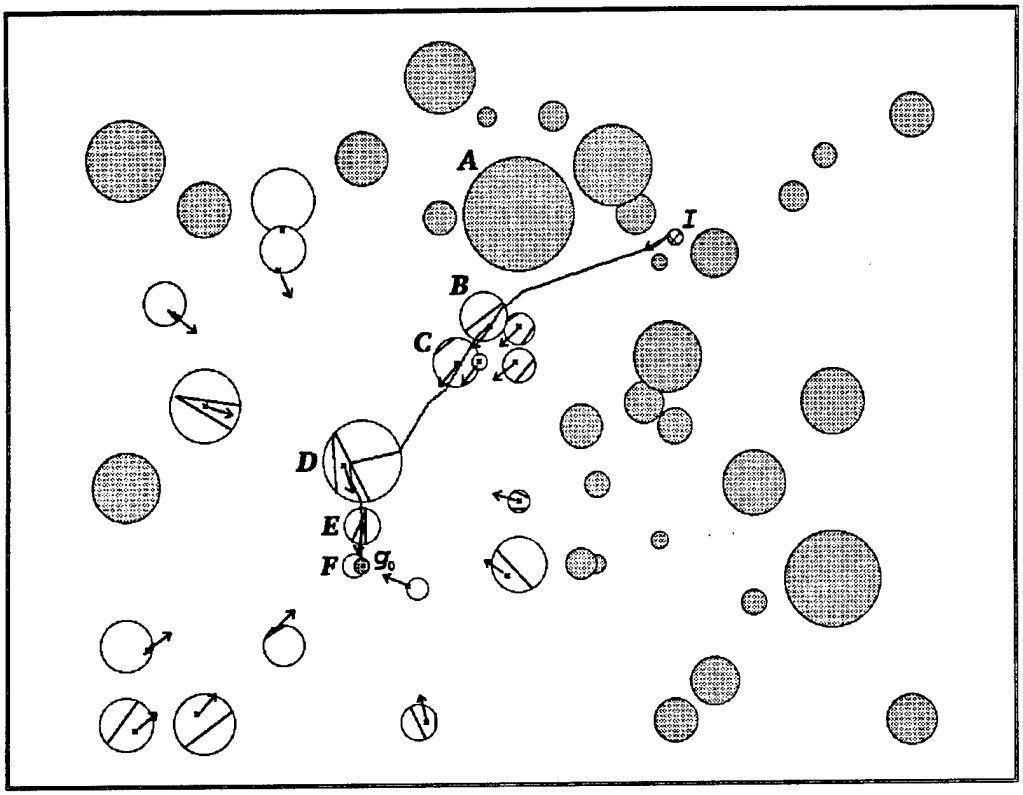


Figure 11: Example with $\theta = 0.2$ radian

L, computes the intersection of L with the directional preimage at the midpoint of each such interval, and keeps the intersection having the biggest area. This intersection is called the *exit region* of L.

(2) Several exit regions, each in a unique disk, may be generated in the same landmark area. With each disk in such a landmark area, the planner associates a P-command leading to one exit region constructed in the area. When several exit regions are available, it selects the region that allows the generation of the P-command containing the smallest number of via points.

The purpose of the first modification is to allow some errors in control and sensing in the landmark areas (see Subsection 9.1). The second modification avoids planning P-commands through long sequences of disks in a landmark area when this is not necessary (however, it is only a heuristic, since it does not take the radii of the landmark disks into account).

Below we present examples of plans generated by the implemented planner, along

with their simulated execution. In all the figures (for instance, see Fig. 11) white disks are landmark disks that intersect the non-directional preimages computed by the planner, except the last one when this last preimage includes the goal region. Grey disks are the other landmark disks; if they have not been touched by the last non-directional preimage computed, no command is attached to them. In all examples, there is a single initial-region disk designated by $\mathcal{I}$, and a single goal-region disk designated by $\mathcal{G}_0$.

Whenever the robot enters a new landmark area L that is part of the termination set of the I-command currently being executed (then the disks in L are necessarily white), it shifts to executing a P-command leading to a point (the exit point) selected in an exit region in L; as soon as it enters the exit region containing the exit point, it abandons the P-command and shifts to executing the corresponding I-command attached to the exit point. The exit region constructed in every white disk, if there is one such region, is shown in the figures (except when it covers the whole disk), together with the direction of the I-command attached to the selected exit point. The termination sets of the I-commands are not shown in the figure, but can be inferred from the drawings.

Fig. 10, 11, and 12 display three examples with the same workspace containing 51 landmark disks of various size and the same initial and goal regions, but with increasing directional uncertainty θ. These examples show that, when uncertainty grows, the planner returns more and more sophisticated plans, as it attempts to reduce uncertainty by leading the robot through additional landmark disks.

In Fig. 10 we set θ to 0.1 radian. The planner returned success after 2 iterations and less than 3 seconds of computation time. Because the directional uncertainty is small, the plan is almost directly aimed toward the goal. The simulated execution produces a path traversing a single landmark disk designated by D before entering the goal kernel. Although the disk marked E is along the path between D and F (the goal's kernel), it is not in the termination set of the I-command executed from the exit point of D. The robot traverses E without shifting to another motion command.

In Fig. 11 we set θ to 0.2 radian. It took 4 iterations of the planner, and 19 seconds of computation, before the initial region was included in a preimage. In the process, the planner attached motion commands to many landmark disks. The simulated execution of the plan produced a path that uses three successive landmark areas designated by B, D, and E, before entering the goal's kernel (F). The area C is also traversed by the path, but it is not part of the termination set of the I-command executed from B.

In Fig. 12 we set θ to 0.3 radian. A plan was generated after 6 iterations, and 52 seconds of computation. A quick comparison of the commanded directions of motion attached to the white landmark disks shows that this plan is quite different

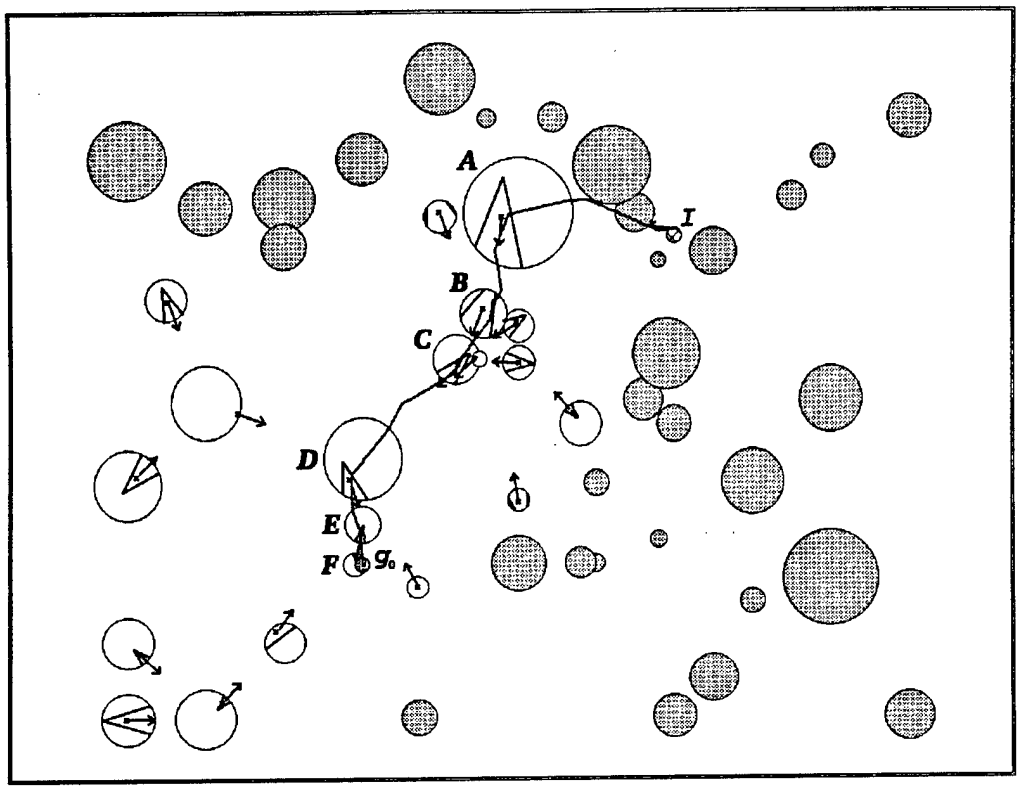
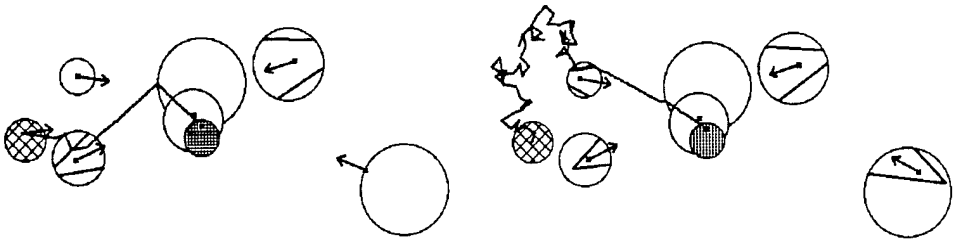


Figure 12: Example with $\theta = 0.3$ radian

from the plan of Fig. 11. The executed path traverses 5 landmark areas designated by A, B, C, D, and E. Notice that both B and C are now used by the navigation system, because it is no longer reliable to directly achieve D from B; C has to be used along the way to reduce uncertainty.

Fig. 13 shows two examples run with another workspace containing 6 landmark disks. In the example on the left, θ was set to 0.3 radian and the planner returned success. In the example on the right, θ was set to 0.35 radian; then no guaranteed plan to the goal exists and the planner returned failure. However, the planner associated motion commands with all landmark disks in the workspace. An R-command was then attached to the initial region. In the sample path shown in the figure, the Brownian subpath resulting from the execution of this command enters the upper-left landmark disk in a relatively short amount of time. From there the navigation system shifts to the safe plan generated by the planner.



$\theta = 0.3$ radian $\theta = 0.35$ radian

Figure 13: Using a Brownian motion to connect to a plan

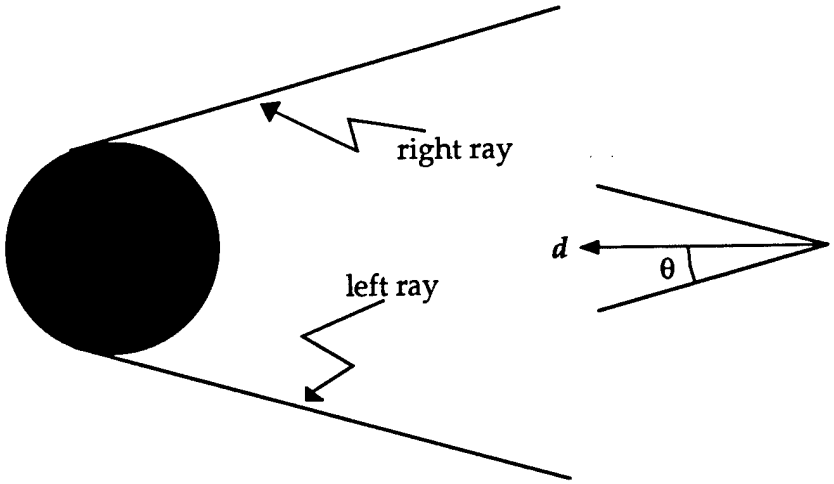


Figure 14: Right and left rays of an obstacle disk

8. Dealing with Obstacles

Let us now introduce $O(\ell)$ forbidden circular regions, the obstacle disks, in the workspace. We assume for simplicity that these disks have null intersection with the landmark and initial-region disks, and do not even touch them. (Retracting this assumption presents no particular difficulty, but significantly increases the number of event types to be considered.)

8.1. Directional Preimage Description

Let us consider an obstacle disk B and a commanded direction of motion d. We define the *right ray* (resp. *left ray*) of B as the half-line tangent to B drawn from

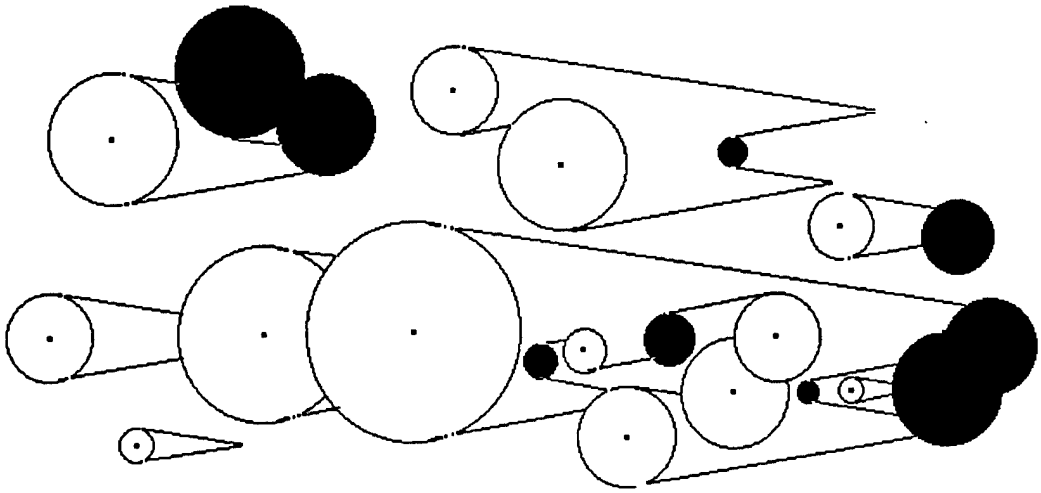


Figure 15: Directional preimage in the presence of obstacle disks

the tangency point in the direction $\pi + d + \theta$ (resp. $\pi + d - \theta$), with the obstacle on its right-hand side (resp. left-hand side), as shown in Fig. 14.

The directional preimage of a goal in the presence of obstacle disks is a region bounded by arcs and edges. Each arc is a subset of the boundary of a kernel or obstacle disk. Each edge is a line segment supported by the right or left ray of a kernel or obstacle disk. When an edge intersects a kernel disk L, the edge is terminated and L is included in the preimage. When an edge intersects an obstacle disk B, the ray is also interrupted, but B is excluded from the preimage. Fig. 15 shows an example of such a directional preimage. Kernel disks are shown white, while obstacle disks are shown black. Notice that the preimage may now contain holes, which themselves may contain components of the preimage. Moreover, its components may have zero, one, or several spikes. The total number of edges and arcs in the preimage's boundary is still in $O(\ell)$.

We construct the description of a directional preimage very much in the same way as when there are no obstacles. We simply give a name to every obstacle disk so that we can label the various edges and arcs of the preimage's boundary that arise from the presence of the obstacles. Since there may be holes, the description of a component of the preimage now consists of nested lists of labels. Although there are more types of labels to handle, the computation of this description still takes time $O(\ell \log \ell)$.

Lemma 6 *The description of a directional preimage in the presence of $O(\ell)$ obstacles is computed in time $O(\ell \log \ell)$.*

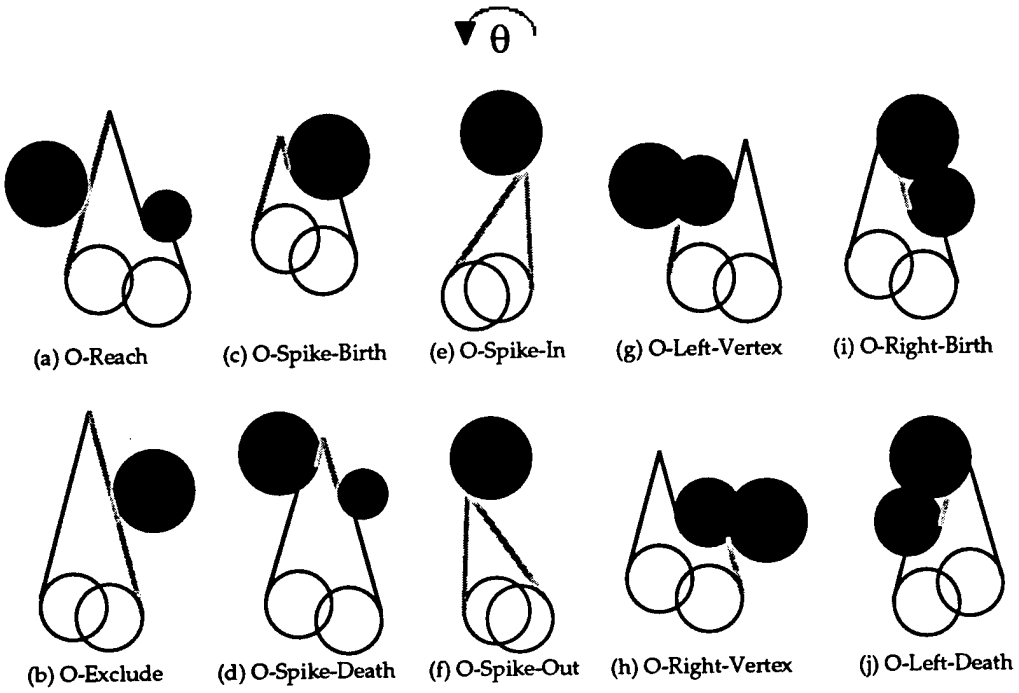


Figure 16: Events caused by obstacle disks

8.2. Critical Directions Due to Obstacles

When the commanded direction of motion d varies over S^1, we have the same types of D-critical events as in Subsection 5.1, plus the following ones, which are caused by obstacle disks (see Fig. 16, where kernel disks are shown white and obstacle disks are shown black):

- An *O-Reach* event occurs when a left edge reaches an obstacle disk by becoming tangent to it.
- An *O-Exclude* event occurs when a right edge leaves an obstacle disk by becoming tangent to it.
- An *O-Spike-Birth* event occurs when a spike emerges as a left edge terminating on an obstacle disk reaches the point where a right edge arises from this disk.
- An *O-Spike-Death* event occurs when a spike vanishes as its left edge, pushed by its right edge, shortens to zero length against an obstacle disk.
- An *O-Spike-In* event occurs when a spike enters an obstacle disk.
- An *O-Spike-Out* event occurs when a spike exits an obstacle disk.
- An *O-Left-Vertex* event occurs when the endpoint of a left edge reaches the inter-

section of two obstacle disks.
- An *O-Right-Vertex* event occurs when the endpoint of a right edge reaches the intersection of two obstacle disks.
- An *O-Right-Birth* event occurs when a right edge emerges at the intersection of two obstacle disks.
- An *O-Left-Death* event occurs when a left edge disappears at the intersection of two obstacle disks.

There are $O(\ell)$ events of types O-Right-Birth and O-Left-Death, $O(\ell^2)$ events of types O-Reach, O-Leave, O-Spike-Birth, O-Spike-Death, O-Right-Vertex, O-Left-Vertex, and $O(\ell^3)$ events of types O-Spike-In and O-Spike-Out.

Lemma 7 *There are $O(\ell^3)$ D-critical directions due to the $O(\ell)$ obstacles.*

8.3. Computation

The directional preimage undergoes a small change at each of the above events. This change is computed in constant time for all events, except O-Spike-In events. At each O-Spike-In event we must identify the label of the arc touched by the spike, which takes $O(\log \ell)$ time. Therefore, the non-directional preimage in the presence of obstacle disks still requires $O(\ell^3 \log \ell)$ to compute, and the time complexity of the planning algorithm remains $O(s\ell^3 \log \ell)$. The space complexity also remains $O(\ell^3)$. Hence:

Theorem 3 *The planning algorithm in the presence of $O(\ell)$ obstacle disks takes time $O(s\ell^3 \log \ell)$ and space $O(\ell^3)$.*

8.4. Experimental Results

Fig. 1 displays both an example of a plan generated by the planner and a sample run of this plan. Obstacle disks are shown black. Landmark disks are shown white or grey depending on whether a reaction rule has been attached to them, or not (see Section 7). The initial region is the disk $\mathcal{I}$. The goal region is the disk $\mathcal{G}_0$. In this example, θ was set to 0.09 radian. The generation of the plan took 12 iterations of the planner, and about 3.5 minutes of computation.

Fig. 17 shows another example with the same landmark and obstacle disks (except for 3 obstacles disks that have been removed) and θ set to 0.1 radian. This example was solved after 6 iterations (the displayed path has only 5 steps) requiring 40 seconds of computation.

Fig. 18 shows a third example with a different workspace containing 34 landmark disks forming 28 landmark areas, and 37 obstacle disks. It was solved in 7 iterations (the displayed path also 7 steps) and less than 6 minutes of computation.

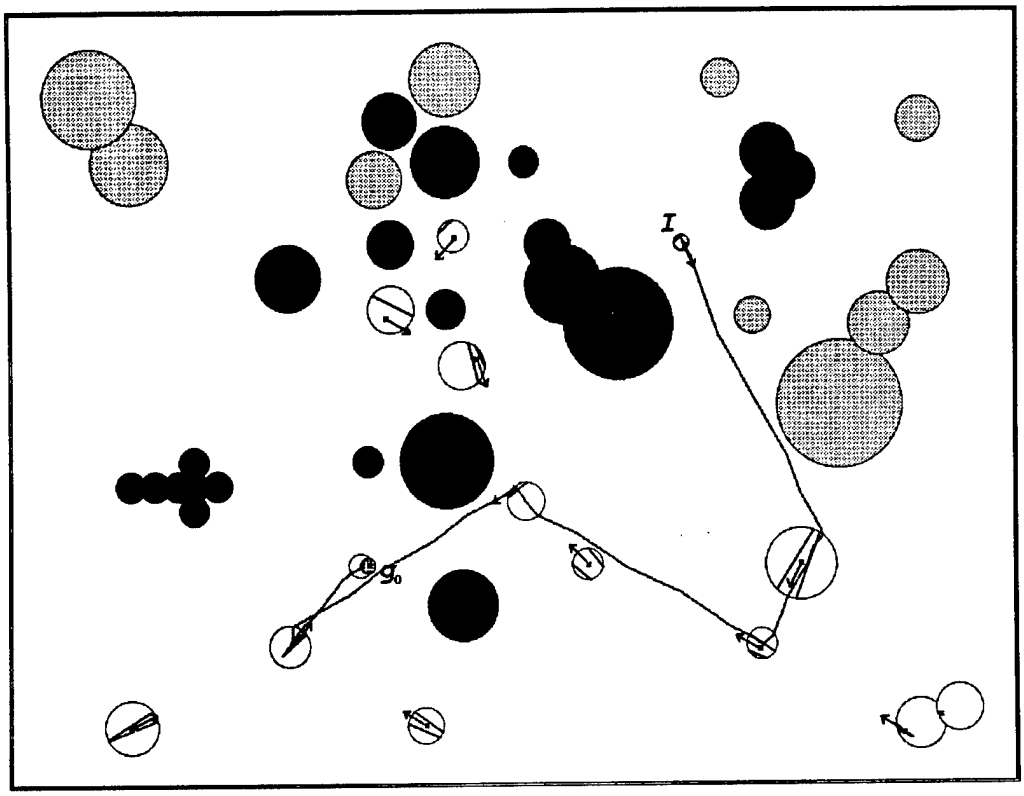


Figure 17: Example with obstacles ($\theta = 0.1$ radian)

Remarks:
- In the presence of obstacles, R-commands (see Subsections 6.2. and 6.3) are allowed to hit obstacles. They generate Brownian motions with reflection on the obstacles' boundary.
- As in the original LMT [26], the planner could easily be adapted to accept compliant I-commands. Such commands would be allowed to hit obstacles, then causing the robot to slide along the obstacles' boundary.

9. Discussion and Extensions

In this section we discuss non-implemented extensions of the planner which eliminate or soften the most restrictive assumptions contained in the problem statement of Section 3.

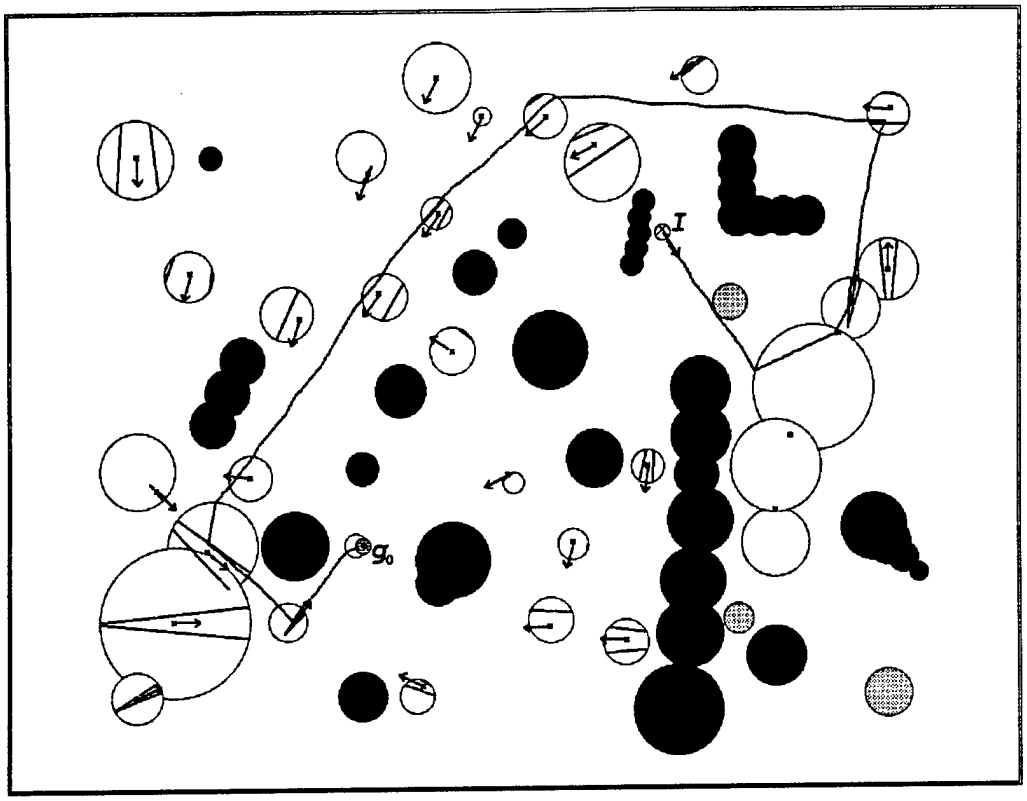


Figure 18: Another example with obstacles ($\theta = 0.1$ radian)

9.1. Uncertainty in Landmark Areas

Perhaps the less realistic assumption in the problem definition of Section 3, with respect to mobile robot navigation, is that control and sensing are perfect in landmark areas, while sensing is null outside any such area. Below, we first argue that this assumption is not too far from reality. We then give the intuition underlying an approach to relieve it.

A typical mobile robot uses two techniques to continuously estimate its position, dead-reckoning and environmental sensing. Environmental sensing provides pertinent information only when some characteristic features of the workspace ("landmarks") are visible by the sensors. Then the robot knows its position with a good accuracy. When no or few features are visible, the robot relies on dead-reckoning, which yields cumulative errors that we model by the directional uncertainty cone. Our assumption that sensing outside landmark areas is null is perhaps conservative, but it does not prevent the robot's navigation system from using all available sensing

information at execution time to better determine the robot's current position. In the worst case this may lead the planner to return failure, while reliable paths exist in practice.

On the contrary, the assumption that control is perfect in the landmark areas is anti-conservative; but if we choose safe features to create landmark disks, it is a reasonable one. To some extent, most workspace can be engineered to include such features. Landmark areas with sharp boundaries can be obtained by introducing artificial landmarks and/or thresholding an estimate of the sensing uncertainty. For example, the notion of a "sensory uncertainty field" (SUF) is introduced in [31, 32]. At every possible point p in the workspace, the SUF estimates the range of possible errors in the sensed position that the navigation system would compute by matching the sensory data against a prior model of the workspace, if the robot was at p. The SUF is computed at planning time from a model of the robot's sensing system.

More interestingly, however, one can notice that perfect control and sensing in landmark areas are not strictly needed. Indeed, once the robot enters a landmark area it is sufficient that it reaches an exit region of non-zero measure prior to executing the next I-command. (If the landmark area intersects the goal, the "exit region" is the intersection with the goal.) For example, the maximal sensing error allowed in a landmark area could be half the radius of the largest disk fully contained in an exit region. Thus, although the planner assumed perfect sensing in landmark areas, we can now create these areas by engineering the workspace in such a way that the sensors just provide the information that is needed by the plan (see [17] for a similar idea).

At the beginning of Section 7, we mentioned a simple way of computing exit regions, but that computation treats landmark disks individually. Other techniques could be developed to compute larger exit regions overlapping several disks in the same landmark area, thus permitting larger errors. In any case, maximal errors in landmark areas seem to depend on the plan itself, so that they can only be computed once a plan has been generated assuming no such errors. We believe, however, that, given a distribution of landmark areas, it is possible to compute a lower bound on the maximal errors that can be allowed in every landmark area over all possible plans. But, in order to turn all these ideas into algorithms, we will have to be more specific and propose an effective model of control and sensing in these areas. We currently work on this issue.

9.2. Landmark and Obstacle Geometry

In our current algorithm, the landmark and obstacle areas are limited to be unions of circular disks. We initially chose to model the fields of influence of the landmarks by disks because we had in mind some sorts of beacons (e.g., radio or infra-red beacons)

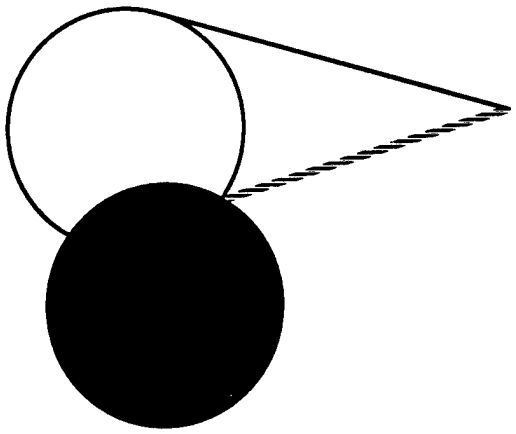


Figure 19: Intersection of a kernel and an obstacle disk

to guide the robot. For simplicity, we made the same choice for the obstacles. However, most natural landmarks do not entail circular fields of influence. We can certainly approximate any landmark or obstacle area by a collection of overlapping disks. However, the number of these disks grows quickly with the precision of the approximation, yielding longer computation.

Fortunately, our algorithm can be easily adapted to deal with landmark and obstacle areas described as generalized polygonal regions bounded by straight and circular edges. In this extension, for any commanded direction of motion, we can still define the right and left rays of a landmark or an obstacle area. If the area is not convex, it may have several right and/or left rays. While the origin of the right or left ray of a circular contour varies continuously as the commanded direction of motion rotates, the origin of the right or left ray of a polygonal contour remains anchored at a fixed vertex, except at critical directions where it jumps from one vertex of the contour to another. These directions (which are parallel to the straight edges of the generalized polygonal contours of the landmark and obstacle areas) are additional critical directions to be treated by the planner. Also, if a kernel landmark area and an obstacle area touch each other, we may have to erect a ray whose origin is an intersection point of the two contours, as shown in Fig. 19 for a kernel disk (shown white) intersecting an obstacle disk (shown black). The origin of such a ray remains stationary for a subrange of orientations d. The curves traced by the spikes are not more complicated than in the pure circular case and their degrees remain no greater than 4.

Although several small adaptations have to be carefully made, our planning method thus extends to the case where landmark and obstacle areas are bounded

by straight edges and circular arcs and may touch each other. If the workspace contains s landmark areas bounded by $O(\ell)$ edges and arcs and the obstacle areas are bounded by $O(\ell)$ edges and arcs, the time complexity of the planner remains $O(s\ell^3 \log \ell)$.

Representing landmark and obstacle areas as generalized polygons is a very realistic model for most applications. In particular, if the robot is an omnidirectional circular robot moving among polygonal obstacles, shrinking the robot to its centerpoint and growing the obstacles isotropically by the robot's radius yields such generalized polygonal regions.

Using algorithms from [12, 3], it is possible to further extend our planning algorithm to include compliant motion commands allowing the robot to slide along obstacle boundaries. This development would be particularly interesting in order to apply our planner to the generation of fine-motion strategies for part-mating operations in assembly tasks. This application would require, however, to allow the "point robot" to move in higher-dimensional spaces, since the configuration space of a moving rigid part in a three-dimensional workspace has dimension 6. In such a space, critical directions of motion are no longer defined as isolated angles, but form submanifolds of various dimensions.

9.3. Varying Directional Uncertainty

The experimental runs with the implemented planner show that the value of θ has major impact on the generated plans. This immediately raises the following question: How to choose θ? In the real world, errors are often difficult to model and not uniformly distributed over an interval. Our argument in favor of the notion of a bounded directional uncertainty is that it makes it possible to define the notion of a guaranteed plan, that is, a plan whose execution is guaranteed to succeed as long as actual errors are within this uncertainty. The advantage of such a plan is that its failures can eventually be traced back to the assumptions made in the problem statement. On the other hand, a too small value of θ may result in plans whose execution is unreliable, while a too large one may produce inefficient plans with too many commands, or no plan at all. One way to deal with this issue is to introduce the notion of a critical value for θ.

Assume that the planner constructs a non-directional preimage $NP(\mathcal{G}_i)$, for some $i \geq 0$, that intersects none of the non-kernel disks. If $NP(\mathcal{G}_i)$ does not contain $\mathcal{I}$, our current planner gives up and returns failure. Then the robot may decide to execute an R-command to connect to the incomplete plan built by the planner.

Instead, rather than giving up, the planner could try to use a smaller value of θ, such that the new non-directional preimage of $\mathcal{G}_i$ either intersects a non-kernel disk, or contains the initial region. In the first case, the planner could resume

planning with the previous value of θ. In the second case, it would return success and a complete plan. In this way, the planner would fail only if a reliable plan does not exist when perfect control ($\theta = 0$) is assumed (this may occur, for example, when there are no landmark areas and the initial region is larger than the goal region). It would always produce a complete plan, but this plan would not be guaranteed. However, a non-guaranteed motion command is more informed than a pure Brownian motion. Hence, by using a plan generated as above, the robot would be more likely to reach the goal quickly than by starting with a Brownian motion. If something goes wrong during plan execution, it will always be possible to turn to a Brownian motion to recover from the incident, as suggested in Subsection 6.3.

How can we compute the new value of θ? Let $P_{\theta_0}(\mathcal{G}_i, d)$ be the directional preimage of $\mathcal{G}_i$ for a direction d, computed with $\theta = \theta_0$. For almost any value θ_0, if we let θ decrease slightly, the edges of the directional preimage rotate continuously, the right edges clockwisely and the left edges counterclockwisely, with the preimage description remaining constant. For isolated values of θ, this is not true. These values are not difficult to identify and compute. Extending this notion to the non-directional preimage of $\mathcal{G}_i$, we say that a value θ_c of θ is *critical* if there exists a direction $d \in S^1$ such that the intersections of $P_{\theta_c+\varepsilon}(\mathcal{G}_i, d)$ and $P_{\theta_c-\varepsilon}(\mathcal{G}_i, d)$, for an arbitrarily small ε, with the initial-region and the non-kernel disks are different. We can now answer the question asked above: The new value of θ should be taken equal to the largest critical value below θ_0.

An exact computation of the critical values of θ is possible by explicitly computing the contour of a non-directional preimage, tracking the variations of this contour when θ decreases, and computing the values of θ when the non-directional preimage intersects a new landmark disk. This computation will be described in a forthcoming publication.

We can extend the use of the critical values of θ further and compute different plans solving the same problem with different values of θ. If a plan generated for some value of θ succeeds without a hitch, this value may be "rewarded"; if, instead, the plan meets multiple unexpected events requiring the execution of random motions, the value may be "punished". By looking at the reward/punish record of the values of θ, the robot may continuously adapt the value of θ to be used by its planner.

10. Conclusion

This paper described a complete polynomial planning algorithm for mobile-robot navigation in the presence of uncertainty. The algorithm addresses a class of problems where landmarks create regions in the workspace where both control and sens-

ing are perfect. Outside these regions sensing is null; control, which relies on dead-reckoning, is imperfect, but directional errors are bounded. Although this class of problems is a simplification of real mobile-robot planning problems, it is by no means oversimplified.

A computer program embedding the planning algorithm was implemented, along with navigation techniques and a robot simulator. This program was run with many different examples, some of which were presented in this paper. The planner is reasonably fast. In its present form, it assumes that all landmark and obstacle areas are described as unions of disks. However, we saw that extending the planner to accept a more general geometry (generalized polygonal areas) is rather straightforward. Other interesting extensions (uncertainty in landmark areas, compliant motion commands, adaptative directional uncertainty) are possible.

So far, most algorithms to plan motion strategies under uncertainty were either exponential in the size of the input problem, or incomplete, or both. Such algorithms may be interesting from a theoretical point of view, but their computational complexity or lack of reliability prevent them from being applied to real-world problems. Our work shows that it is possible to identify a restricted, but still realistic, subclass of planning problems that can be solved in polynomial time. This subclass is obtained through assumptions whose satisfaction may require prior engineering of the robot and/or its workspace. In our case, this implies the creation of adequate landmarks, either by taking advantage of the natural features of the workspace, or introducing artificial beacons, or using specific sensors. We call this type of simplification *engineering for planning tractability.*

Engineering the robot and the workspace has its own cost and we would like to minimize it. Therefore, future research should aim at finding more general classes of problems than the one solved by the current planner, but requiring less engineering and still solvable in polynomial time. It should also investigate the following "inverse" problem: Given our planning method and the description of a family of tasks (e.g., the set of all possible initial and goal regions), how to minimally engineer the workspace, e.g., what is the minimal number of landmarks that we should place in the workspace and where should we place them, so that every possible problem admits a guaranteed solution.

References

[1] Anderson, R.F. and Orey, S., "Small Random Perturbations of Dynamical Systems with Reflecting Boundary," *Nagoya Math. J.*, 60, 1976, pp. 189-216.

[2] Ayache, N., Artificial Vision for Mobile Robots: Stereo Vision and Multisensory Perception, The MIT Press, Cambridge, MA, 1991.

[3] Briggs, A.J., An efficient Algorithm for One-Step Planar Compliant Motion Planning with Uncertainty, Technical Report, Department of Computer Science, Cornell University, Ithaca, NY, 1988.

[4] Buckley, S.J., Planning and Teaching Compliant Motion Strategies, Ph.D. Dissertation, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, 1986.

[5] Canny, J.F. and Reif, J., "New Lower Bound Techniques for Robot Motion Planning Problems," *27th IEEE Symposium on Foundations of Computer Science*, Los Angeles, CA, 1987, pp. 49-60.

[6] Canny, J.F., The Complexity of Robot Motion Planning, MIT Press, Cambridge, MA, 1988.

[7] Canny, J.F., "On Computability of Fine Motion Plans," *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Scottsdale, AZ, 1989, pp. 177-182.

[8] Chapman, D. and Agre, P.E., "Abstract Reasoning as Emergent from Concrete Activity," *Reasoning about Actions and Plans*, edited by Georgeff, M.P. and Lansky, A.L, Morgan Kaufmann Publishers, Los Altos, CA, 1986, pp. 411-424.

[9] Christiansen, A., Mason, M. and Mitchell, T.M., "Learning Reliable Manipulation Strategies without Initial Physical Models," *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Cincinnati, OH, 1990, pp. 1224-1230.

[10] Crowley, J.L., "World Modeling and Position Estimation for a Mobile Robot Using Ultrasonic Ranging," *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Scottsdale, AZ, 1989, pp. 674-680.

[11] Donald, B.R., "A Geometric Approach to Error Detection and Recovery for Robot Motion Planning with Uncertainty," *Artificial Intelligence J.*, 37(1-3), 1988, pp. 223-271.

[12] Donald, B.R., The Complexity of Planar Compliant Motion Planning Under Uncertainty, Tech. Rep. 87-889, Dept. of Computer Science, Cornell University, 1987.

[13] Donald, B.R. and Jennings, J., "Sensor Interpretation and Task-Directed Planning Using Perceptual Equivalence Classes," *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Sacramento, CA, 1991, pp. 190-197.

[14] Drummond, M. "Situated Control Rules," *Proc. of the First Int. Conf. on Principles of Knowledge Representation and Reasoning*, Morhan Kaufmann Publishers, Los Altos, CA, 1989, pp. 103-113.

[15] Erdmann, M., On Motion Planning with Uncertainty, Tech. Rep. 810, AI Lab., MIT, Cambridge, MA, 1984.

[16] Erdmann, M., On Probabilistic Strategies for Robot Tasks, Ph.D. Dissertation, Tech. Rep. 1155, AI Lab., MIT, Cambridge, MA, 1990.

[17] Erdmann, M., Towards Task-Level Planning: Action-Based Sensor Design, Tech. Rep. CMU-CS-92-116, Dept. of Computer Science, Carnegie Mellon University, Pittsburgh, PA, February 1992.

[18] Goldberg, K.Y., Stochastic Plans for Robotic Manipulation, Ph.D. Dissertation, Rep. CMU-CS-90-161, Dept. of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1990.

[19] Friedman, J., Computational Aspects of Compliant Motion Planning, Ph.D. Dissertation, Report No. STAN-CS-91-1368, Dept. of Computer Science, Stanford University, Stanford, CA, 1991.

[20] Guibas, L.J. and Stolfi, J., Ruler, Compass, and Computer: The Design and Analysis of Geometric Algorithms, Tech. Rep. No. 37, Digital, Systems Research Center, Palo Alto, 1989.

[21] Hutchinson, S., "Exploiting Visual Constraints in Robot Motion Planning," *Proc. of the IEEE Int. Conf. of Robotics and Automation*, Sacramento, CA, 1991, pp. 1722-1727.

[22] Latombe, J.C., Robot Motion Planning, Kluwer Academic Publishers, Boston, MA, 1991.

[23] Latombe, J.C., Lazanas, A., and Shekhar, S., "Robot Motion Planning with Uncertainty in Control and Sensing," *Artificial Intelligence J.*, 52(1), 1991, pp. 1-47.

[24] Leonard, J.J. and Durrant-Whyte, H.F., "Mobile Robot Localization by Tracking Geometric Beacons," *IEEE Tr. on Robotics and Automation*, 7(3), 1991, pp. 376-382.

[25] Levitt, T.S., Lawton, D.T., Chelberg, D.M. and Nelson, P.C., "Qualitative Navigation," *Image Understanding Workshop*, Los Angeles, CA, 1987, pp. 447-465.

[26] Lozano-Pérez, T., Mason, M.T. and Taylor, R.H., "Automatic Synthesis of Fine-Motion Strategies for Robots," *Int. J. of Robotics Research*, 3(1), 1984, pp. 3-24.

[27] Mahadevan, S. and Connell, J., Automatic Programming of Behavior-based Robots using Reinforcement Learning, Research Rep., IBM T.J. Watson Research Center, Yorktown Heights, NY, 1990.

[28] Mason, M.T., "Automatic Planning of Fine Motions: Correctness and Completeness," *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Atlanta, GA, 1984, pp. 492-503.

[29] Preparata, F.P. and Shamos, M.I., Computational Geometry: An Introduction, Springer Verlag, New York, 1985.

[30] Schoppers, M.J., Representation and Automatic Synthesis of Reaction Plans, Ph.D. Dissertation, Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, 1989.

[31] Takeda, H. and Latombe, J.C., "Sensory Uncertainty Field for Mobile Robot Navigation," *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Nice, France, 1992.

[32] Takeda, H. and Latombe, J.C., Planning the Motions of a Mobile Robot in a Sensory Uncertainty Field, Tech. Rep., Dept. of Computer Science, Stanford University, Stanford, CA, 1992.

APPENDIX: COMPUTATION OF SPIKE EVENTS

In this appendix we establish the equation of the locus of a spike and the intersection of this locus with a circle. These results are needed to schedule potential spike events (I-Spike-In, I-Spike-Out, L-Spike-In, L-Spike-Out, O-Spike-In, O-Spike-Out). They may also be used to compute critical values of θ (see Subsection 9.3).

A. SPIKE-LOCUS CURVE

A spike is the intersection point of two rays parallel to the sides of the directional uncertainty cone and tangent to two disks, which may, or may not, be distinct. Their angle is constant and equal to 2θ, with its bisector oriented along the commanded direction of motion d. We are interested in the equation of the locus of the spike as d varies in S^1.

Let us consider the case where the intersecting rays are tangent to two distinct landmark disks L_1 and L_2 of respective radii η_1 and η_2. One ray, l_1, is the left ray of L_1; the other, r_2, is the right ray of L_2, as shown in Fig. 20. The results established below remain valid when any of the disks is an obstacle disk, provided that we change the corresponding radius η_i ($i = 1$ or 2) into $-\eta_i$ (Section D will provide more detail). If the spike is produced by a single landmark disk, then its locus is simply a circle having the same center as the landmark disk; its radius is $\eta/\sin\theta$, where η is the radius of the disk.

Let φ, φ_1 and φ_2 denote the angles between the x-axis of a workspace coordinate system and the direction d, the ray l_1, and the ray r_2, respectively. We have $\varphi_1 = \varphi - \theta$ and $\varphi_2 = \varphi + \theta$. Let the center of the disk L_1 (resp. L_2) be c_1 (resp. c_2) with coordinates (x_1, y_1) (resp. (x_2, y_2)). We let p_1 (resp. p_2) denote the origin of l_1

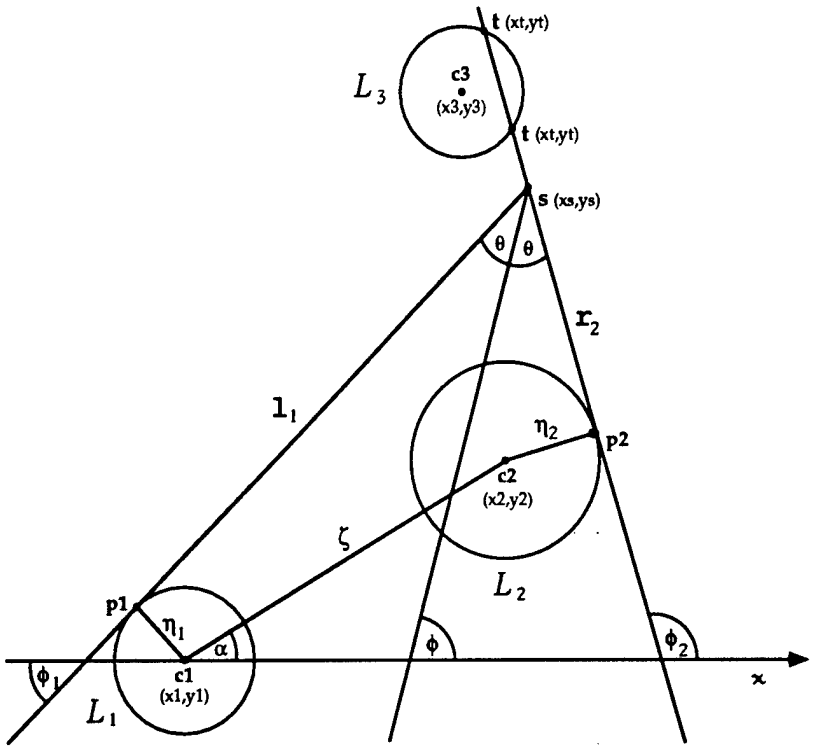


Figure 20: A spike created by two landmark disks

(resp. r_2) and s denote the intersection point of l_1 and r_2, i.e. the point we wish to track. Let (x_s, y_s) be the coordinates of s in the workspace coordinate system.

The coordinates of p_1 are $(x_1 - \eta_1 \sin\varphi_1,\ y_1 + \eta_1 \cos\varphi_1)$, and those of p_2 are $(x_2 + \eta_2 \sin\varphi_2,\ y_2 - \eta_2 \cos\varphi_2)$. Hence, the equations for l_1 and r_2 are:

$$\begin{aligned} l_1 :&\ -(x - x_1)\sin\varphi_1 + (y - y_1)\cos\varphi_1 - \eta_1 = 0, \\ r_2 :&\ -(x - x_2)\sin\varphi_2 + (y - y_2)\cos\varphi_2 + \eta_2 = 0. \end{aligned}$$

Both equations must be verified for $x = x_s$ and $y = y_s$, yielding the following parametric equations of the spike-locus curve:

$$\begin{aligned} x_s &= \frac{1}{\sin 2\theta}[\eta_1 \cos\varphi_2 + \eta_2 \cos\varphi_1 - x_1 \cos\varphi_2 \sin\varphi_1 + \\ &\qquad x_2 \cos\varphi_1 \sin\varphi_2 + (y_1 - y_2)\cos\varphi_1 \cos\varphi_2], \qquad (1) \\ y_s &= \frac{1}{\sin 2\theta}[\eta_1 \sin\varphi_2 + \eta_2 \sin\varphi_1 + y_1 \cos\varphi_1 \sin\varphi_2 - \\ &\qquad y_2 \cos\varphi_2 \sin\varphi_1 - (x_1 - x_2)\sin\varphi_1 \sin\varphi_2]. \qquad (2) \end{aligned}$$

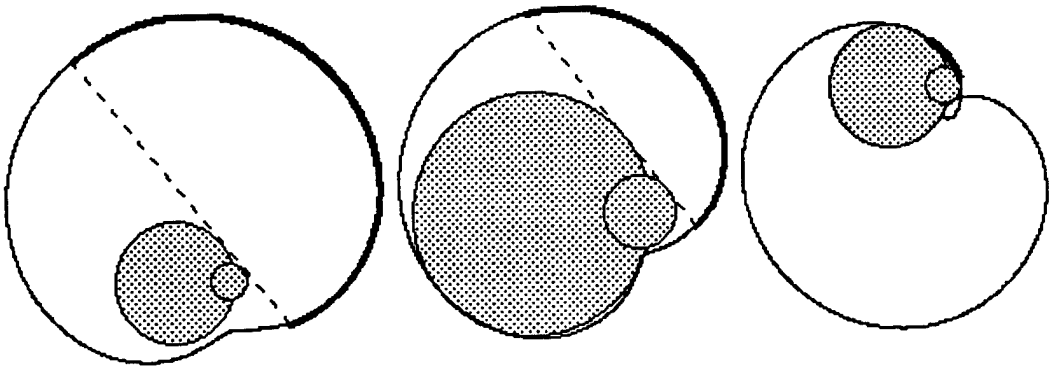


Figure 21: Various shapes of spike-locus curves

Since both φ_1 and φ_2 are linear functions of φ, both x_s and y_s are thus expressed as functions of φ. Eliminating φ from Equ. (1) and (2) yields a fourth-degree equation representing the locus of s. However, we will see that the above parametric form suffices.

Different shapes of the spike-locus curve are possible, which depend on the signs of the quantities $\lambda_1 = |\eta_1 \cos 2\theta + \eta_2| - \zeta$ and $\lambda_2 = |\eta_2 \cos 2\theta + \eta_1| - \zeta$, where ζ denotes the distance between c_1 and c_2. These shapes are illustrated in Fig. 21, for two intersecting landmark disks, with L_1 being the biggest of the two disks:
- If both λ_1 and λ_2 are positive, the locus is a simple (Jordan) curve that encloses the two landmark disks without touching any of them.
- If λ_1 or λ_2 is positive and the other is negative, the locus is still a simple curve, but it is twice tangent to L_1.
- If both λ_1 and λ_2 are negative, the curve is no longer simple; it makes a loop and is twice tangent to both disks.
(In the first example of Fig. 21, $\eta_1 = 50$, $\eta_2 = 15$, and $\theta = .25$; both λ_1 and λ_2 are positive. In the second example, $\eta_1 = 100$, $\eta_2 = 30$, and $\theta = .65$; λ_1 is negative and λ_2 positive. In the third example, $\eta_1 = 50$, $\eta_2 = 15$, and $\theta = 1.37$; both λ_1 and λ_2 are negative.) In the case where the two disks do not intersect, the quatities λ_1 and λ_2 are always both negative; the spike-locus curve makes a loop and is twice tangent to both disks, as shown in Fig. 22.

Notice that not all points in a spike-locus curve correspond to feasible spikes. Let us draw the line tangent to both L_1 and L_2, and oriented so that it touches L_1 before L_2 (the dashed line in Fig. 21). The valid part of the locus (shown in thicker line in the figure) lies on the left-hand side of this line. Therefore, intersecting a spike-locus curve with a circle may yield both valid and invalid points, which we must classify afterwards.

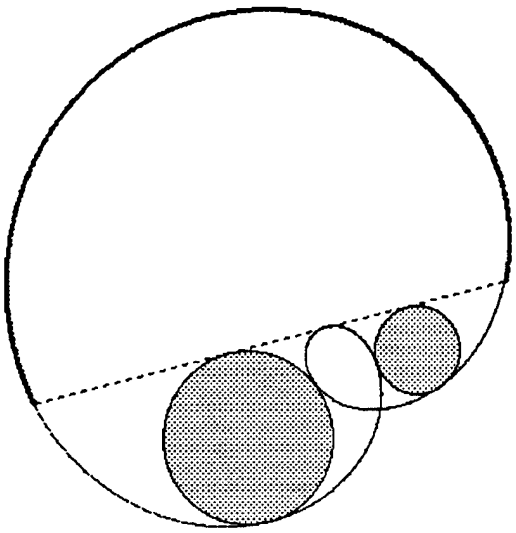


Figure 22: Spike-locus curve for two disjoint landmark disks

B. Intersection of a Spike-Locus Curve with a Circle

Let us now consider a third disk centered at c_3 and having radius η_3 (see Fig. 20). We wish to compute the intersection of the spike-locus curve with the circle C bounding this disk. To that end, we denote any intersection point of r_2 with C by t, compute the vectors $s-p_2$ and $t-p_2$ as functions of φ, and solve the equation $s-p_2 = t-p_2$ for φ.

We have $s - p_2 = (x_s - x_2 - \eta_2 \sin\varphi_2\ ,\ y_s - y_2 + \eta_2 \cos\varphi_2)$. Using Equ. (1) and (2), we get (after some calculation):

$$s - p_2 \quad = \quad \frac{A}{\sin 2\theta}(\cos\varphi_2\ ,\ \sin\varphi_2), \tag{3}$$

where $A = (y_1 - y_2)\cos\varphi_1 - (x_1 - x_2)\sin\varphi_1 + \eta_1 + \eta_2 \cos 2\theta$.

Let the equation of the circle C be:

$$(x - x_3)^2 + (y - y_3)^2 = \eta_3.$$

By solving it together with the equation of r_2, we get the coordinates (x_t, y_t) of the intersection t of r_2 with C. After yet some calculation, we find:

$$\begin{aligned} x_t \quad = \quad & x_3 \cos^2\varphi_2 + x_2 \sin^2\varphi_2 + \eta_2 \sin\varphi_2 - \\ & (y_2 - y_3)\sin\varphi_2 \cos\varphi_2 \pm \cos\varphi_2 \sqrt{\eta_3^2 - B^2}, \end{aligned} \tag{4}$$

$$y_t = (x_2 + x_3)\cos\varphi_2 \sin\varphi_2 - R_2 \cos\varphi_2 + y_3 \sin^2\varphi_2 + y_2 \cos^2\varphi_2 \pm \sin\varphi_2 \sqrt{\eta_3^2 - B^2}, \quad (5)$$

where $B = (y_2 - y_3)\cos\varphi_2 - (x_2 - x_3)\sin\varphi_2 - \eta_2$.

We have $t - p_2 = (x_t - x_2 - \eta_2 \sin\varphi_2\ ,\ y_t - y_2 + \eta_2 \cos\varphi_2)$. Using Equ. (4) and (5), we get:

$$t - p_2 = (C \pm \sqrt{\eta_3^2 - B^2})\ (\cos\varphi_2\ ,\ \sin\varphi_2), \quad (6)$$

where $C = (x_3 - x_2)\cos\varphi_2 + (y_3 - y_2)\sin\varphi_2$.

Comparing Equ. (3) and (6) we see that the equality of the components of the two vectors yields the same equation, namely:

$$A = (C \pm \sqrt{\eta_3^2 - B^2})\ \sin 2\theta.$$

After rearranging to isolate the root, squaring, and performing a considerable amount of calculation we end up with:

$$S_0 + S_1 \sin\varphi + S_2 \cos\varphi + S_3 \sin^2\varphi + S_4 \cos^2\varphi + S_5 \sin\varphi\cos\varphi = 0, \quad (7)$$

where:

$$\begin{aligned}
S_0 &= (x_3^2 + y_3^2 - \eta_3^2)\sin^2 2\theta + \eta_1^2 + \eta_2^2 + 2\eta_1\eta_2 \cos 2\theta, \\
S_1 &= 2\eta_1(A_1 + A_2 \cos 2\theta - C_1 \sin 2\theta) + 2\eta_2(A_2 + A_1 \cos 2\theta - C_2 \sin 2\theta), \\
S_2 &= 2\eta_1(B_1 + B_2 \cos 2\theta - D_1 \sin 2\theta) + 2\eta_2(B_2 + B_1 \cos 2\theta - D_2 \sin 2\theta), \\
S_3 &= A_1^2 + A_2^2 + 2A_1A_2 \cos 2\theta - 2(A_1C_1 + A_2C_2)\sin 2\theta, \\
S_4 &= B_1^2 + B_2^2 + 2B_1B_2 \cos 2\theta - 2(B_1D_1 + B_2D_2)\sin 2\theta, \\
S_5 &= 2[A_1B_1 + A_2B_2 + (A_1B_2 + A_2B_1)\cos 2\theta - \\
&\qquad (A_1D_1 + B_1C_1 + A_2D_2 + B_2C_2)\sin 2\theta],
\end{aligned}$$

and

$$\begin{aligned}
A_1 &= -x_1 \cos\theta + y_1 \sin\theta, & A_2 &= x_2 \cos\theta + y_2 \sin\theta, \\
B_1 &= x_1 \sin\theta + y_1 \cos\theta, & B_2 &= x_2 \sin\theta - y_2 \cos\theta, \\
C_1 &= -x_3 \sin\theta + y_3 \cos\theta, & C_2 &= x_3 \sin\theta + y_3 \cos\theta, \\
D_1 &= x_3 \cos\theta + y_3 \sin\theta, & D_2 &= x_3 \cos\theta - y_3 \sin\theta.
\end{aligned}$$

Using the transformation $u = \tan(\varphi/2)$, Equ. (7) becomes:

$$(S_0 - S_2 + S_4)u^4 + 2(S_1 - S_5)u^3 + 2(S_0 + 2S_3 - S_4)u^2 + 2(S_1 + S_5)u + (S_0 + S_2 + S_4) = 0 \quad (8)$$

which is a fourth-degree equation that can be solved analytically. From u, we compute φ, and from it (x_s, y_s) using (1) and (2).

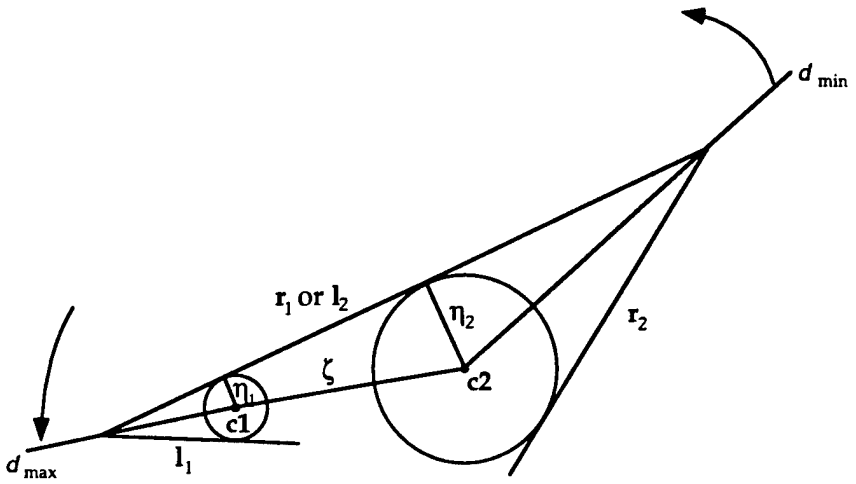


Figure 23: Spike valid limits

C. Classification of Solutions

As we mentioned above, not every real solution of Equ. (8) corresponds to a feasible spike. We still have to disqualify invalid solutions. We also need to classify the valid solutions into *entry* and *exit* angles when d varies counterclockwisely.

A spike-locus curve can have up to four intersection points with a circle. However, a spike is feasible only for values of φ in the interval $[\varphi_{min}, \varphi_{max}]$, where φ_{min} (resp. φ_{max}) is the angle between the x-axis and the direction d when l_1 (resp. r_2) is the exterior common tangent to both L_1 and L_2 (see Fig. 23). Denoting the distance between c_1 and c_2 by ζ, and the angle between the x-axis and the vector $c_2 - c_1$ by α, we have:

$$\begin{aligned} \varphi_{min} &= \alpha + \arcsin \frac{\eta_2 - \eta_1}{\zeta} + \theta, \\ \varphi_{max} &= \pi + \alpha + \arcsin \frac{\eta_2 - \eta_1}{\zeta} - \theta. \end{aligned}$$

Any solution $\varphi \in [\varphi_{min}, \varphi_{max}]$ is either an entry angle (i.e., the spike enters the disk) or an exit angle (i.e., the spike exits the disk), or both (i.e., the spike-locus curve is tangent to the circle bounding the disk). Let $\varphi_s = \arctan(y'_s/x'_s)$, with $x'_s = dx_s/d\varphi$ and $y'_s = dy_s/d\varphi$, and $\varphi_d = \arctan((x - x_3)/(y_3 - y))$ be the angles of the x-axis with the tangents to the spike locus and the circle C, respectively, at their intersection point. A solution φ is an entry angle if $\varphi_s \in (\varphi_d, \varphi_d + \pi)$ and an exit angle if $\varphi_s \in (\varphi_d + \pi, \varphi_d + 2\pi)$. If $\varphi_s = \varphi_d + 2\pi$, the spike locus and C are tangent and exterior to each other (see Fig. 24 (a)). If $\varphi_s = \varphi_d$ the two curves are

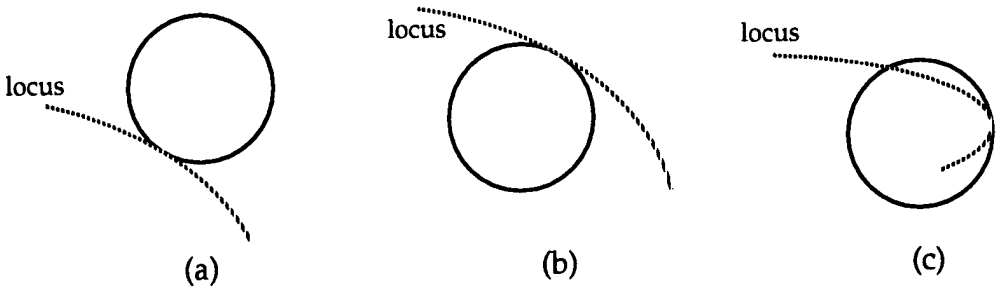


Figure 24: Different tangent positions

tangent with one of them lying inside the other (see Fig. 24 (a) and (b)). Let η_s be the radius of curvature of the spike locus at the point of tangency. We have:

$$\eta_s = \frac{[(x_s')^2 + (y_s')^2]^{3/2}}{|y_s'' x_s' - x_s'' y_s'|},$$

with $x_s'' = d^2x_s/d\varphi^2$ and $y_s'' = d^2y_s/d\varphi^2$. If $\eta_s > \eta_3$ the spike locus encloses C (Fig. 24 (b)); if $\eta_s < \eta_d$ the spike locus is enclosed by C (Fig. 24 (c)); if they are equal we need higher derivative tie-breakers which are too tedious to mention here. (Actually, in the main body of this paper, we assume that the disks are in general position, so that no two critical events occur simultaneously. Therefore, we may ignore the cases where the spike-locus curve is tangent to C. However, the study of this case is of interest if we wish to compute the critical values of θ where the envelop of a non-directional preimage becomes tangent to a disk, as suggested in Subsection 9.3.)

D. Spikes with Obstacle Rays

Spikes involving rays arising from obstacle disks are slightly different. Since obstacles must be avoided, a left ray leaves the obstacle on its right, and a right ray on its left (see Fig. 14). Fortunately, the only difference in the spike equations established above is a change of sign. If a ray of a spike arises from an obstacle disk of radius η_i ($i = 1$ or 2), we just need to change η_i into $-\eta_i$ wherever it appears in the equations.

However, the range of φ where a spike is feasible merits some discussion. Let us consider a spike whose left and right rays stem from a landmark disk and an obstacle disk, respectively. Assume for the moment that the two disks do not intersect. When φ varies (as d spans S^1), this spike emerges from the obstacle disk, with its right ray tangent to the obstacle at this same point. Therefore, the valid subset of the spike locus begins exactly at a point where the spike-locus curve and the obstacle

disk intersect or are tangent. A straightforward calculation shows that:

$$\varphi_{min} = \alpha + \arcsin \frac{\eta_2 \cos 2\theta - \eta_1}{\zeta} + \theta.$$

The spike terminates when its right ray becomes the internal common tangent of the two disks. Hence:

$$\varphi_{max} = \pi + \alpha - \arcsin \frac{\eta_2 + \eta_1}{\zeta} - \theta.$$

In a similar fashion, we can calculate the limits for an obstacle-landmark spike:

$$\varphi_{min} = \alpha + \arcsin \frac{\eta_2 + \eta_1}{\zeta} + \theta,$$
$$\varphi_{max} = \pi + \alpha + \arcsin \frac{\eta_2 - \eta_1 \cos 2\theta}{\zeta} - \theta,$$

and for an obstacle-obstacle spike:

$$\varphi_{min} = \alpha + \arcsin \frac{\eta_2 \cos 2\theta + \eta_1}{\zeta} + \theta,$$
$$\varphi_{max} = \pi + \alpha - \arcsin \frac{\eta_2 + \eta_1 \cos 2\theta}{\zeta} - \theta.$$

Regarding the shape of the spike-locus curve, it still sepends on the sign of the two quantities λ_1 and λ_2 defined above, by substituting $-\eta_i$ for η_i whenever we refer to an obstacle disk.

When the two disks do not intersect, both quantities $|\pm\eta_1 \cos 2\theta \pm \eta_2| - \zeta$ and $|\pm\eta_2 \cos 2\theta \pm \eta_1| - \zeta$ are always negative. Then the spike-locus curve always contains an inner loop (see Fig. 25).

Let us consider now the case when the two disks intersect each other. (In the main body of this paper, we accept intersecting obstacle disks, but assumed for simplification that obstacle disks were disjoint from landmark disks. The following shows that the case where obstacle and landmark disks intersect each other is not difficult to handle, provided that we introduce additional events in the computation of the non-directional preimages.) Again we begin by examining a landmark-obstacle spike. (See Fig. 26 for illustration.) The termination of this spike cannot occur at the internal tangent point, since this point no longer exists. When the origin of the right ray reaches the intersection point of the two disks (an *O-Right-Stick* event), it sticks there for a while as the ray continues to rotate counterclockwisely. When the ray gets tangent to the landmark disk (an *O-Right-Release* event), it turns into a landmark ray and its origin starts moving in the boundary of the landmark disk.

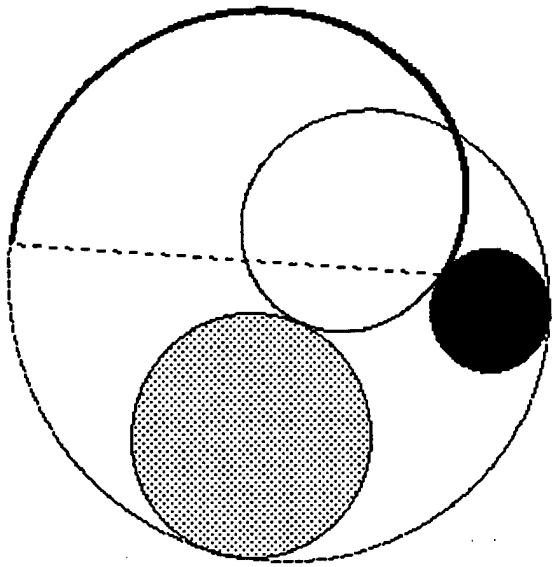


Figure 25: Spike-locus curve for disjoint landmark and obstacle disks

Thus, the landmark-obstacle spike was transformed, first into a *right-stuck spike*, and then into a landmark-landmark spike. Since the right-stuck spike has a different equation, the termination of the original landmark-obstacle spike occurs exactly at the O-Right-Stick event. The calculation of the O-Right-Stick angle yields:

$$\begin{aligned} \varphi_{min} &= \alpha + \arcsin \frac{\eta_2 \cos 2\theta - \eta_1}{\zeta} + \theta, \\ \varphi_{max} &= \pi/2 + \alpha - \arccos \frac{\eta_2^2 + \zeta^2 - \eta_1^2}{2\zeta\eta_2} - \theta. \end{aligned}$$

The landmark-obstacle spike exists if and only if $\varphi_{min} < \varphi_{max}$, which translates into the following constraint:

$$\eta_1^2 + \eta_2^2 - \zeta^2 \quad < \quad 2\eta_1\eta_2 \cos 2\theta. \tag{9}$$

This same constraint guarantees the existence of an obstacle-landmark spike with:

$$\begin{aligned} \varphi_{min} &= \pi/2 + \alpha + \arccos \frac{\eta_1^2 + \zeta^2 - \eta_2^2}{2\zeta\eta_1} + \theta, \\ \varphi_{max} &= \pi + \alpha + \arcsin \frac{\eta_2 - \eta_1 \cos 2\theta}{\zeta} + \theta. \end{aligned}$$

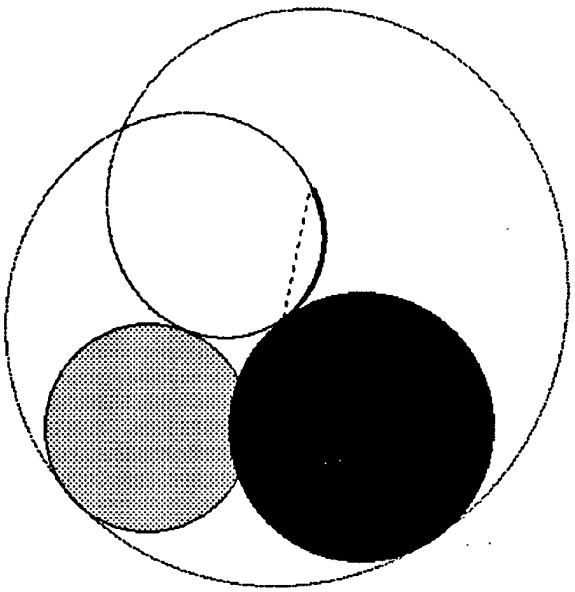


Figure 26: Spike-locus curve for intersecting landmark and obstacle disks

Two intersecting obstacle disks may create a spike, but this spike can only occur "under" the obstacles (see Fig. 27). The constraint for its existence is that the (exterior) angle of the tangents to the two disks at an intersection point be less than 2θ. This translates into the constraint:

$$\zeta^2 - \eta_1^2 - \eta_2^2 \;>\; 2\eta_1\eta_2 \cos 2\theta. \tag{10}$$

The valid range of φ is the same as in the non-intersecting case.

Our final point will be to prove that whenever an obstacle disk is involved in a spike, the spike-locus curve includes a loop. We already know that this is true for two disjoint disks. So we only consider the case of two intersecting disks. First, $\eta_1 \geq \eta_1 \cos 2\theta$ implies $\eta_1 - \eta_2 \geq \eta_1 \cos 2\theta - \eta_2$. Since the disks intersect, we have $\zeta > \eta_1 - \eta_2$, thus:

$$\zeta \;>\; \eta_1 \cos 2\theta - \eta_2. \tag{11}$$

On the other hand, the constraint (9) for the existence of the spike implies:

$$\frac{\zeta^2 + \eta_2^2 - \eta_1^2}{2\eta_2\zeta} \;>\; \frac{\eta_2 - \eta_1 \cos 2\theta}{\zeta}.$$

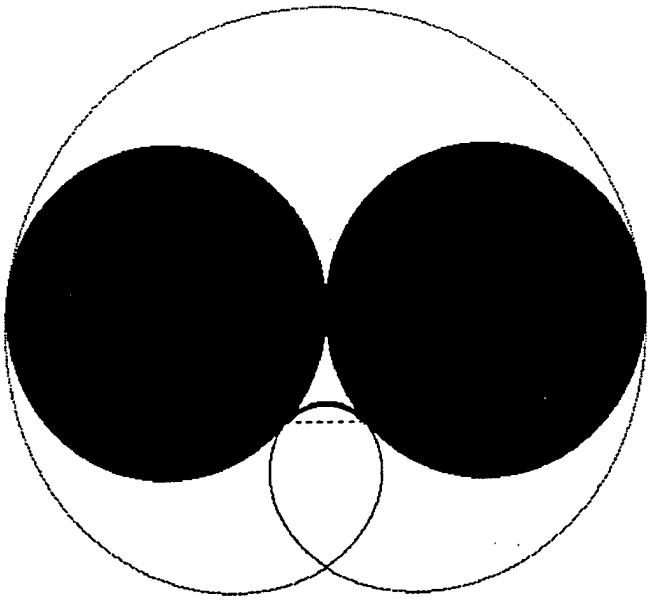


Figure 27: Spike-locus curve for two intersecting obstacle disks

The left-hand side of this inequality is equal to the cosine of the angle between the segment c_2c_1 and the segment joining c_2 to an intersection point of the circles bounding the two disks. So, it is less than one, which yields

$$\zeta \; > \; \eta_2 - \eta_1 \cos 2\theta. \tag{12}$$

Combining the relations (11) and (12), we get $\zeta \geq |\eta_2 - \eta_1 \cos 2\theta|$, which yields $\lambda_1 \leq 0$ for the obstacle-landmark and landmark-obstacle spikes. For the obstacle-obstacle spike we can also prove that $\lambda_1 \leq 0$ starting from equation (10) and the fact that $\eta_1 \geq -\eta_1 \cos 2\theta$, and working in a similar as above fashion. In a symmetric way, we also get $\lambda_2 \leq 0$ in all cases, proving that when at least one obstacle is involved, the spike-locus curve always contains a loop when there exists a valid range of values of φ.