REASONING ABOUT UNCERTAINTY IN ROBOT MOTION PLANNING

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Anthony Lazanas
August 1994

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Jean-Claude Latombe
(Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Leonidas J. Guibas

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Oussama Khatib

Approved for the University Committee on Graduate Studies:

Dean of Graduate Studies

Abstract

Robot motion planning has been a central problem of robotics research since its early stages, yet no satisfactory solution has been found to date. The difficulty stems from the fact that the real world is too complex to be modelled accurately. The details that are omitted from a model create uncertainty. More detailed models have less uncertainty, but increase computational complexity; less detailed models may result in incorrect and/or incomplete planners. In this thesis, we investigate the effects of uncertainty on the difficulty of robot motion planning, and we study the tradeoff between physical and computational complexity.

Uncertainty makes the execution of motion plans non-deterministic. Sensing is used to help the robot identify its state, but measurement errors interfere with the process of state identification. Reasoning about potential (actual and perceived) states during plan execution is what makes planning with uncertainty a difficult problem. To keep the complexity of the problem polynomial, we have to represent all possible states with a limited number of equivalence classes. We present a model of a robot and its workspace, which yields a polynomial, correct, and complete motion planning algorithm. The key idea is the existence of regions in the workspace (landmark regions), where the robot has perfect position sensing and can navigate without error. Outside the landmark regions, the robot has no sensing at all and navigates with bounded control error. The proposed algorithm creates a distributed motion plan, by associating with every landmark region a motion command. This command is guaranteed to bring the robot into the goal, or into another region with a similar motion command attached to it.

Planning is performed using the preimage backchaining method. We extend the standard definition of a "nondirectional preimage" (the set of all points of the workspace from where a robot can reliably achieve a goal set) to the case where a motion command depends on an arbitrary number of control parameters. We show that the resulting multi-dimensional preimage can be represented with a polynomial number of 2-D slices, each computed for a critical combination of values of the parameters. We present implemented algorithms for one parameter (the commanded direction of motion) and for two parameters (the commanded direction of motion and the directional uncertainty). The latter algorithm has many interesting applications, like planning in anisotropic uncertainty environments, finding probabilistic plans when no guaranteed plans exist, avoiding unexpected obstacles, etc.

Experimentation with the algorithm using a real mobile robot has been successful. By engineering the workspace, we have been able to satisfy all the assumptions of our planning model. As a result, the robot can operate for long periods of time with no failures. Since our planning algorithms are provably correct, experimental failures can be attributed to violations of the assumptions on which the algorithm depends. Recursive workspace engineering can be used until all assumptions are satisfied. Thus, experimentation and workspace engineering become essential tools for the creation of useful robot motion planning algorithms.

Acknowledgements

When writing, the author feels (and it has to be so) immense loneliness. So, it is with impatience that I reach this section, eager to reunite (at least spiritually) with all those people who have contributed one way or another to the completion of my dissertation.

My advisor, Jean-Claude Latombe, has been able to create an environment at the Robotics Lab that combines academic excellence with camaraderie, hard work with a relaxed atmosphere. The quality in him that I appreciate the most, is his willingness to be always accessible (his office door is always open), his eagerness to get involved in long discussions, ready to listen, process and enrich the ideas presented to him.

I am grateful to my readers, Jean-Claude, Leo Guibas, and Oussama Khatib who found the time to go through the length of this work, and help me identify and rectify its weaknesses.

My research has been generously supported by IBM (Manufacturing Fellowship 1988), DARPA (contract DAA21-89-C0002) and ONR (contract N00014-92-J-1809). Thanks to everybody who believed in my work and supported it.

The Robotics Lab community, with top-caliber people performing research extending from Algorithms, Computational Geometry, Artificial Intelligence and Computer Vision, to Control, to Mechanical and Electrical Engineering, has helped broaden my understanding of robotics, and has made me aware of the difficulties and the details involved when building a complete autonomous system. Especially, I thank my colleagues, Tsai-Yen, Yotto, Mark, Lydia, Randy, Wonyon, David, Shashank, Dan, and everyone else, for their enthusiasm and their readiness to share good and difficult moments.

I am deeply indebted to David, Philippe, Ben, Ken and Craig, the people who helped me complete the experiments with NOMAD. Without their zeal and hard work it wouldn't have been possible.

I would also like to thank Mike Erdmann and Bruce Donald for the inspiration I drew from their work, as well as for their suggestions and answers to my questions.

All my friends at Stanford made my life in California so enjoyable, that these years remain in my memory as a long happy day full of one-to-one basketball games (thank you, George, for helping me build such a good record) and melodic gatherings (thank you, Michael, for singing so nicely). I also thank all my friends away from Stanford, for staying so close despite the distance.

Achilles, Lydia, Michael and Effie deserve my special gratitude, the first for meticulously proof-reading my dissertation, Lydia for being my personal representative at Stanford while I was away and for spending so much time getting signatures and printing my thesis, and Michael and Effie for all their help with the final details.

Finally, I thank my parents for setting the example and teaching me hard work, honesty, and the joy of life. Also, for enduring my prolonged absence; I hope now that "the book is over," we will be able to see each other more often.

My brother and sister-in-law, besides everything else, for giving me the best dissertation completion present.

And Vasso, for everything.

Contents

List of Figures

Chapter 1

Introduction

In dealing with physical reality, which is too complex to be accurately modelled, one must decide on an appropriate modeling granularity level. The more details in the model, the bigger the complexity, and the harder the problems become. If too many details are omitted, then the class of solvable problems becomes uninteresting. Once a model of the real world has been selected, everything that has been omitted from the model becomes *uncertainty*. Dealing with uncertainty is the major problem of most systems aspiring to operate in the real world, and has been the most significant obstacle for the introduction of robots in everyday life. This work addresses the problem of dealing with uncertainty in robot motion planning. The general ideas presented, however, are applicable to any system that deals with the physical world.

1.1 About robots, motions, and planning

A *robot* is an autonomous device that is capable of changing the physical state of the world in which it operates, according to some guidelines specified by an external operator. The level of autonomy of a robot is determined by the level of details that it requires for its operation. For example, a robot at the human level of autonomy would accept commands expressed in ordinary human language; a superhuman robot would accept a single command, "Satisfy my wishes!," and would act accordingly. Since robots are neither human nor superhuman, guidelines must be expressed in an appropriately simple language.

As opposed to information, which requires very low energy and can be represented and propagated with electromagnetic interaction, a physical change requires a relatively high energy exchange. The simplest mechanism to deliver a burst of energy at a specific location is contact. Contact requires motion, so most operations of a robot involve some kind of motion. The study of robotic behavior is, to a large extent, the study of the motion of mechanical devices.

Every task, expressed in some input language that a robot accepts, must be translated into a collection of motions. Such a collection is called a *motion plan.* The procedure of deriving motion plans from more abstract task specifications is not an easy task for humans, so we require that robots perform it automatically. Two modules are usually involved in this process. First, a *task planner* translates commands expressed in a human-like language (e.g., "Get me a glass of water") into a set of geometric specifications (e.g., "Go to the refrigerator, grab the door, open the door, ..."). Then, a *motion planner* translates the high-level geometric specifications into a set of primitive motions, i.e., motions that can be easily effected with simple control of the motors that drive the robot.

The physical state of a robot and its environment can be described by a vector of parameters $\vec{\mathbf{p}}$. The geometric specifications of tasks can be described by a set of constraints $\mathcal{G}(\vec{\mathbf{p}})$. The initial state of the system can be described by another set of constraints $\mathcal{I}(\vec{\mathbf{p}})$. Motion planning amounts to the construction of an algorithm of primitive motions that transform $\mathcal{I}(\vec{\mathbf{p}})$ into $\mathcal{G}(\vec{\mathbf{p}})$, given some description of the effects of primitive motions on $\vec{\mathbf{p}}$. This formulation indicates that robot motion planning is just an instance of the more general problem of controlling the behavior of a system that obeys certain rules. Therefore, most of the ideas that we develop here have a much bigger domain of potential applications.

1.2 About uncertainty

In a perfect world, the result of primitive motions is deterministic. In the real world, however, nothing is deterministic. After all, the Second Coming may occur any time. Introducing uncertainty to the outcome of primitive motions complicates motion planning considerably. To begin with, a motion plan may not be a simple string of primitive motions any more. Since a primitive motion may lead the robot to a number of different states, which motion must be executed next may be a function of the resulting state. This observation introduces the need to sense the state of the system after the execution of each primitive motion. The sensing mechanism introduces its own errors: The identification of states may not be perfect. Because of that, a motion-planning algorithm must reason about the information that may be revealed after each primitive motion, as well as about the reliability of this information. As uncertainty grows, the number of states that may result from a primitive motion also grows, making the task of a motion planner more complex.

The deepest questions regarding uncertainty are philosophical in nature. Is it possible to predict the future given complete knowledge of the present? Is there such thing as complete knowledge? What constitutes knowledge? The first two questions are equivalent to the question regarding the existence of God, to which no satisfactory answer has be given to date. Adopting a utilitarian point of view, it is possible to give an answer to the third question. Knowledge is a collection of "words" in some representation language. Representation is the concept underlying knowledge. So the question regarding complete knowledge becomes a question about complete representation. Are there languages rich enough to represent the physical world? Even if we assume that such a language exists, we must use measurement in order to identify the actual state of the world in the space defined by the language. Measurement is subject to the limitations of measuring devices, and subject to Heisenberg's uncertainty principle. Finally, there is always the question whether the real world is a deterministic system or not, i.e., whether there exists some divine randomness.

Regardless of philosophical views, uncertainty is a major hindrance in the attempt to control physical systems. Its existence must be acknowledged and dealt with. Various approaches can be used in dealing with uncertainty.

- **Ignoring:** The simplest approach is to proceed as if uncertainty were not present and converge to the goal state with iterative process execution. The advantage of this method is that it is easy to design and execute. The disadvantages are (a) that it may often fail, (b) that, even when successful, it may take a big number of iterations, and (c) that it is hard to explain failure and derive useful conclusions from it.

- **Engineering:** Another approach is to attempt to eliminate as many sources of uncertainty as possible by simplifying the structure of the physical system and aligning it with the capabilities of the measuring devices. Care must be taken that the engineering of the system do not interfere with its ability to perform its assigned duties.
- **Reducing:** A third approach is to use a richer representation language. The more detailed the description of a system, the less uncertain its operation is. However, using a rich representation language increases computational complexity and may make problem solving hard.
- **Reasoning:** Uncertainty that has not been ignored, eliminated by engineering, or reduced by representation, becomes a parameter of the system model. To predict the behavior of the system we must explicitly reason about uncertainty.

1.3 Problem solving methodology

Problem solving in the real world has four phases. The first phase (*engineering*) is an attempt to reduce the physical complexity of a given problem, by means of physical alteration of the characteristics of the problem. Irrelevant details are eliminated, helpful features are added, and mechanisms are devised and implemented, so that the original problem becomes easier to solve. During the second phase (*modeling*), the simplified actual problem is transformed into a formal problem through the choice of a representation language and a solution language. The formal problem is the projection of the real problem onto the space defined by the representation and the solution language. The physical complexity of the real problem is transformed into the computational complexity of the formal problem. The choice of the representation and the solution language is a major factor in determining the computational complexity of the formal problem. During the third phase (*computation*), a mapping between the representation language space and the solution language space is defined

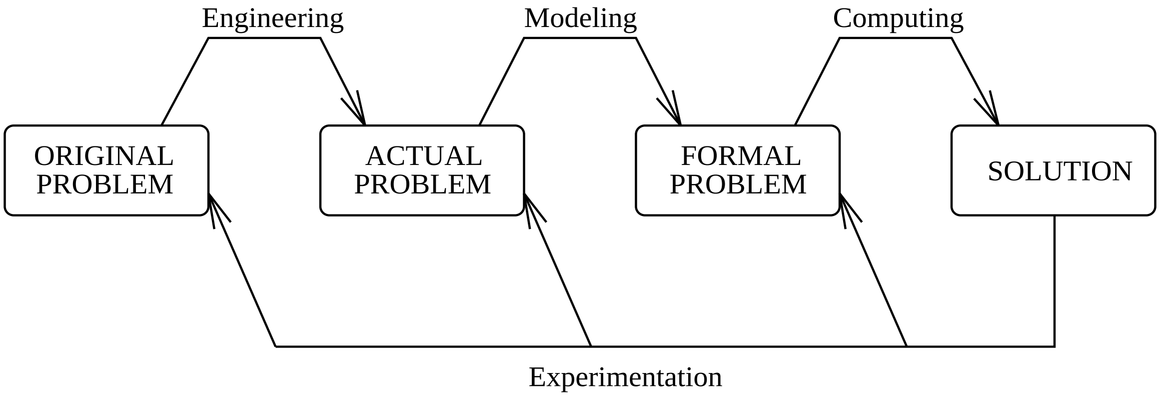


Figure 1.1: Problem solving phases

as the solution of the formal problem. An algorithm is a procedure that attempts to compute the above mapping. Finally, *experimentation* is used to evaluate the quality of a problem solver. Failures are attributed to deficiencies of engineering, modeling, or computation, which are accordingly adjusted to increase the effectiveness of the problem solver.

The quality of a problem solver is evaluated on three grounds: *correctness*, *completeness*, and *complexity*. Correctness measures the ability of the problem solver to produce solutions which are actual solutions of the original problem. Completeness measures the percentage of actual solutions which can be produced by the problem solver. Complexity is judged by how fast the solver generates the solutions, as a function of the size of the input problem.

Even though the above attributes can be associated with all phases of problem solving, they are only used to evaluate the computation phase. The reason is that it is easier to quantify the attributes using the description language of the formal problem. The computational complexity of a formal problem is defined as the lowest possible complexity of a correct and complete algorithm that solves the problem. The computational complexity of a formal problem depends on the physical complexity of the underlying physical problem, as well as on the representation and solution language.

The term "epistemic adequacy" has been used to describe how well a formal problem approximates a real one. The choice of the representation and the solution language determine the epistemic adequacy of modeling. Although high epistemic adequacy is desirable, it usually results in formal problems that cannot be solved with algorithms that are correct, complete, and of low complexity. In dealing with hard problems, an important decision to be made is how many physical details of the original problem we are willing to hide through engineering and modeling, in order to produce a formal problem with sufficiently low complexity.

1.4 The philosophy of the proposed approach

Robot motion planning in the real world is a difficult problem mainly because of the existence of uncertainty. Two major schools of thought have evolved. The "reactionists" ignore uncertainty, with the claim that it is too expensive to reason about it. Motion planning is performed as if no uncertainty were present. Robots are equipped with a set of predefined local behaviors to help them react to the eventual failures during plan execution. The "reasoners" model uncertainty (or some part of it) and devise algorithms that reason explicitly about it, so plans do not fail as often. However, the models that have been used to describe uncertainty result in intractable formal problems. To date, no universally acceptable solution has been given to the problem of robot motion planning with uncertainty. Algorithms are either too slow or unacceptably incorrect.

The role of engineering Both approaches ignore the value of problem engineering as a means to reduce the complexity of the motion-planning problem. In this work, we begin from the premise that we need motion-planning algorithms which are correct, complete, and have computational complexity that is a polynomial function of the size of the formal problem. We use workspace and robot engineering to produce a real problem that can be modeled into a formal problem, which in turn can be solved with algorithms having the above properties. In our approach, engineering is not confined to creating robots able to cope with a given environment, but it attempts to adapt the environment to the capabilities of robots (much in the same way that paved roads are constructed to help cars navigate). Therefore, our approach is geared towards applications in environments that can be controlled (at least to some extent), and not towards purely exploratory applications.

The role of experimentation By insisting on the correctness of algorithms, we are able to redefine the role of experimentation. When an algorithm is provably correct, experimental failures can be directly attributed to the violation of one or more modeling assumptions. Then, we can decide whether we can augment the used model (always retaining the desired properties of the algorithm), or whether further engineering is necessary. Thus, workspace engineering enters the problem-solving loop, as experimental feedback can be used to provide guidelines for engineering.

Modeling requirements The correctness of a motion planning algorithm means that the plans it produces are guaranteed to succeed. A motion plan can be guaranteed in the presence of uncertainty only if uncertainty is bounded. If anything can happen, no plan is guaranteed to succeed. Therefore, we represent stochastic quantities with distributions that are bounded over some domain. Since we are only interested in the correctness of a plan (and not in its stochastic attributes), we do not reason at all about the characteristics of the probability distributions; rather, we only use their bounded domains as a sufficient representation.

To solve the motion-planning with uncertainty problem in polynomial time, it is necessary that the planner only consider a small finite number of world states. We embed such a discrete structure into the robot workspace by introducing uniquely identifiable characteristics called *landmarks*. Each state is characterized by the perception of a unique landmark. All other information is shielded out of the planning model to preserve its polynomial complexity. Each landmark is associated with a region of the workspace (a landmark region) which allows a robot to navigate in it without loss of information. Uncertainty increases during navigation outside such regions and decreases when the robot enters one of them. Therefore, the role of the planner is to make sure that every motion of the robot will eventually terminate within some landmark region. Such a design allows the robot to navigate while keeping its uncertainty bounded. This approach has distinct similarities to the binary transmission of signals, where a limited representation language (0 and 1) is used to preserve the quality of data.

1.5 Overview of the thesis

In Chapter 2 we present a very general model of a system that is controllable through a set of primitive commands, and that is capable of affecting the physical state of the world in which it operates. We begin with a very simple example, and then we gradually build on it introducing many important issues and structures. We study planning under uncertainty, and we show how we can derive guaranteed plans by bounding uncertainty over some domain. We discuss perception and interpretation of perceived data, and we demonstrate how a planner must explicitly reason about information that may become available at execution time. The more information that is available, the more complicated the planning task is. Therefore, it is of utmost importance to select the right pieces of information: those pieces that enable a planner to solve most tasks in an efficient manner. Too much information may be unnecessary, too little information may be inadequate. We give several versions of planning and execution algorithms. Since it is difficult for a planner to produce plans that react to the sensed data at a very fast rate, we decouple the motion control loop from the plan-execution control loop and we introduce the notion of continuously executed commands and the termination condition. Finally, we present high-level definitions of the forward projection, the preimage, and the kernel, as tools for producing guaranteed plans under uncertainty.

In Chapter 3 we present the general problem of robot motion planning and, more specifically, robot motion planning under uncertainty. After an overview of the literature in the field, we introduce the notion of the configuration space and formally define the planning problem. Then we present the preimage backchaining planning technique and an implementable version in two dimensions, backprojection from a target kernel. We consider three kinds of sensing, position sensing, force sensing and sticking on obstacle edges, and we show that their combined use may have a nonlinear effect on the capabilities of a planner. Several actual planning examples, solved with the implemented planner are presented. We also present an algorithm to compute the maximal preimage of a convex set, and we show that a concave set may have no unique maximal preimage. By combining two-dimensional preimages computed for all possible commanded directions of motion, we create three-dimensional omnidirectional and nondirectional preimages; these are essential building blocks of complete polynomial planners under uncertainty.

In Chapter 4 we discuss in detail how the amount of information considered by a planner affects the complexity of the planning algorithm. Practical planners must be fast and, therefore, must select carefully what information to consider. Since the appropriate information may not always be available in the workspace, we propose the idea of workspace engineering for planning complexity reduction. We introduce the notion of landmarks as sources of information, and the notion of landmark regions as information fields in the workspace. By assuming a finite number of such fields, we make the first step towards polynomial planning. We give a formal definition of the problem of landmark-based navigation and a correct, complete, and polynomial algorithm for it. The algorithm is based on the fact that a nondirectional preimage can be represented with a data structure of polynomial complexity. This polynomial description consists of a collection of slices (two-dimensional preimages) computed at certain critical values of the commanded direction of motion d. The critical events that produce the critical values of d are listed and explained in detail. We present several examples of the algorithm. At the end of the chapter, we explain why the motion plans produced by our planning algorithm have many desirable properties. Specifically, in addition to having been produced by a correct, complete, and polynomial algorithm, the motion plans are optimal in the number of executed steps, distributed over the workspace, and robust (i.e., it is easy to reattach to a plan that has been disrupted by an unexpected event).

In Chapter 5 we discuss several shortcomings of the landmark-based navigation algorithm and we attempt to overcome them. We show that it is straightforward (a) to allow generalized polygonal objects (instead of simple disks) in the configuration space, (b) to consider compliant motions, and (c) to allow contact between landmark regions and C-obstacles. We also show that it is easy to precompute all possible plans in a specific workspace. The assumption that is the hardest (actually impossible) to satisfy is that sensing and control are perfect within landmark areas. To deal with this issue, we introduce the notion of generalized landmarks, where the robot is only required to be able to get reliably within a subregion of the landmark region. Planning with generalized landmarks is actually much easier than planning with our standard definition of landmarks. We then attempt to relax the constraint that all landmarks must be recognizable from each other, and present techniques that deal with confusable landmarks. We conclude that confusable landmarks introduce too many complications and, in general, lead to loss of polynomiality. Therefore, making landmarks individually recognizable is an important goal of workspace engineering. Finally, we give a graph representation of planning with uncertainty. We are able to represent with a graph all possible plans, including guaranteed plans, plans that always fail, and plans that may fail or succeed. An important family of plans are the probabilistically guaranteed plans (introduced by Erdmann in [34]), which are plans that are guaranteed to succeed, but the number of execution steps is not bounded. (The expected number of execution steps, though, is a finite number.) Many problems that do not accept guaranteed plans accept probabilistically guaranteed plans. We give a recursive algorithm to compute such plans within our landmark framework, but we make no attempt to analyze its complexity.

Up to this point we have assumed that the robot controls a single parameter, the commanded direction of motion d. In Chapter 6 we consider the situation where more than one parameters are changeable, either under the control of the robot, or not. In this case, the two-dimensional preimage depends on many parameters, and the notion of nondirectional preimage is generalized to the multi-parametric preimage. We show that the ideas of Chapter 5 can be easily extended to higher-dimensional spaces. As a specific case, we study planning where the directional uncertainty half-angle θ is controllable by the robot. We wish to produce guaranteed plans using the highest possible values of θ, since, presumably, some kind of cost must be paid in order to reduce uncertainty. We develop a correct, complete, polynomial, maximum-uncertainty one-step planner, as well as a multi-step one that maximizes the value of θ at each step. We show that these algorithms can be also used in the case where θ

is not controllable by the robot, for problems that do not accept guaranteed plans. In this case, the algorithms produce non-guaranteed plans that have the maximum likelihood of success. The algorithms have been implemented and their use is demonstrated with several examples. Because non-guaranteed plans may fail in undesirable ways, we explore ways to produce plans that either succeed or fail recognizably (Donald's EDR plans [23]). Finally, we present many practical problems that can be solved using the multi-parametric planning technique (planning with anisotropic directional uncertainty, unexpected obstacle avoidance, controllable sensor sensitivity, robot cooperation, and directional sensing).

In Chapter 7 we test our algorithms by experimenting with them on a real robot. We use NOMAD-200, a small omnidirectional cylindrical robot from Nomadic Technologies. We conduct two sets of experiments. In the first set we use prismatic landmarks standing in the workspace that are detected by the reflection of a plane of laser light on them. In the second set of experiments we use landmarks placed on the ceiling that are detected by a camera fixed on the robot and looking straight up. We describe in detail the computation of the induced landmark regions, and the detection, recognition and localization algorithms. In the first set of experiments, not all of our model assumptions are met and the execution of plans is not 100% reliable. In the second set of experiments, we use iterative workspace engineering (especially with respect to the location of landmarks) to eliminate as many sources of errors as possible. Since we cannot completely eliminate uncertainty in the landmark regions, we make use of the generalized landmarks algorithm. The result is a system that operates reliably until the robot runs out of power (a little less than an hour). This is possible because in our system errors do not deteriorate over time, but always remain within their assumed bounds. We conclude that, with relatively little effort and resources, it is possible to create a planning system that remains reliable over time.

In the last chapter, we summarize the conclusions from this work, we identify promising areas for future research, and we discuss questions raised by critics of our approach.

Chapter 2

Models of Perception and Action

Before we can solve problems in the physical world, we must first build an adequate representation of it. This representation, called a *world model*, is a projection of the physical world onto the reasoning space. The projection must maintain all important characteristics of the world (relative to the problem we wish to solve), but it should conceal all parameters that are irrelevant to the problem. Selecting an appropriate world model greatly affects the existence and the complexity of problem solving algorithms.

In this chapter we consider a very generic dynamic world model. An agent (robot), obedient to commands from a controlling system, operates in the world. The agent acts as the interface between the controlling system and the world, by providing the means to affect the world state, and also by providing sensory feedback. We begin with a very simple discrete model with no uncertainty. Then we consider uncertainty both about the present (initial state uncertainty, perceptual uncertainty) and the future (control uncertainty). We introduce several functions that facilitate planning with uncertainty, and we sketch a planning algorithm. Finally, we consider systems with two distinct control loops, the motion control loop and the plan execution control loop, and we introduce the notion of the "termination condition."

2.1 Action

Consider a robot $\mathcal{R}$ operating in a workspace $\mathcal{W}$. Let the vector s contain all the parameters that are needed to describe the combination of the robot and the workspace. We will refer to s as the *state* of the system. Let c denote any *primitive command* that the robot can execute. The function **act** describes the results of the execution of c, i.e., it describes how s changes in response to the execution of c. Assume for the moment that this is the only way in which s may change, and that the system is completely deterministic.

Let s_0 represent the initial state of the system. A *plan* is a sequence of commands $c_0, c_1, \ldots, c_{n-1}$, which, if executed, will cause the system to pass through states $s_0, s_1, \ldots, s_{n-1}$ and finally arrive at s_n, where $s_i = \mathbf{act}(c_{i-1}, s_{i-1})$, $i = 1$ to n. Plans are provided by a user[1] trying to achieve goals with the help of the robot. A goal is defined as a set $\mathbf{G}$ of desirable final states of the system. *Planning* is the procedure which takes as inputs s_0, $\mathbf{G}$ and **act** and returns a sequence of primitive commands, i.e., a plan. The trace of an execution of a plan is the trajectory of states $s_0, s_1, \ldots, s_n$ that the system traverses during this execution. An execution is successful if $s_n \in \mathbf{G}$.

As an example, consider a point robot that can move on a straight line. Let the set of primitive commands that are available to the user be $\{F, B\}$, and let the function **act** be defined as follows:

$$\mathbf{act}(c, s) = \begin{cases} s + 1, & \text{if } c = F; \\ s - 1, & \text{if } c = B. \end{cases}$$

If $s_0 = 0$ and $\mathbf{G} = [2.5, 5]$, then plans (F, F, F) and (F, F, F, F) both achieve the goal, the first one in three steps and the second one in four. Generally, plans with fewer steps are preferable, unless there are other considerations like robustness. For example, plan (F, F, F, F) has a margin of error equal to 1, and might be preferable to plan (F, F, F) which has a 0.5 margin of error.

---

[1] A person, another robot, or a software program.

2.2 Uncertainty

Planning in deterministic systems is straightforward, though not always easy. Unfortunately, real systems are never that well behaved. There is always a stochastic aspect in their behavior. We call this stochastic aspect *uncertainty.* While in fully deterministic systems a plan has a unique execution trace, this is not the case in stochastic systems. We call a plan *guaranteed* if its executions are always successful. The probability of success of a non-guaranteed plan is the percentage of its successful executions.

Uncertainty about the initial state: Let us first consider the case where the initial state is not precisely known at planning time. Thus, instead of a single value s_0, the planning procedure knows a probability density function $f_0(s)$ about the possible initial states of the system. We introduce a new function **ACT** that takes $f_i(s)$ as input (instead of s), and returns the updated distribution of s after the execution of c_i: $f_{i+1}(s) = \mathbf{ACT}(c_i, f_i(s))$. Now the success probability of a plan is equal to $\int_{\mathbf{G}} f_n(s)\, ds$.

Returning to our simple example, assume that s_0 is not known precisely, but it is known to be somewhere in [-0.6,0.4] with uniform probability. If $u[a,b]$ denotes the uniform distribution on [a,b], then one possible form of **ACT** is the following:

$$\mathbf{ACT}(c, u[a,b]) = \begin{cases} u[a+1, b+1], & \text{if } c = F; \\ u[a-1, b-1], & \text{if } c = B. \end{cases}$$

With the goal being [2.5,5] the plan (F, F, F), is not guaranteed. It maps $u[-0.6, 0.4]$ into $u[2.4, 3.4]$, so its probability of success is 90%. However, the plan (F, F, F, F) is guaranteed.

Control Uncertainty: As long as **act** is deterministic, the variance of $f_i(s)$ remains constant and equal to the variance of $f_0(s)$ throughout the execution of a plan. In reality, however, this is rarely true. The mapping **act** is stochastic, causing the variance of $f_i(s)$ to grow as i grows. We call this fact *control uncertainty.*

In our example, let us assume for simplicity that **ACT** retains the uniformity of the distribution, but stretches it a bit, like for example:

$$\mathbf{ACT}(c, u[a,b]) = \begin{cases} u[a+0.8, b+1.2], & \text{if } c = F; \\ u[a-1.2, b-0.8], & \text{if } c = B. \end{cases}$$

Plan (F, F, F) maps the initial distribution $u[-0.6, 0.4]$ into $u[1.8, 4]$; its probability of success is equal to 15/22. But now, plan (F, F, F, F) also becomes non-guaranteed, because it results in a final distribution of $u[2.6, 5.2]$ that is not fully included in the goal; its probability of success is equal to 24/26.

2.3 Perception

In the above example, one could note that the plan (F, F, F, F) is guaranteed to make the robot *reach* the goal, but not to make the robot *stop* in the goal. Indeed, after three steps the distribution of s is $u[1.8, 4]$. If $s \leq 3.8$ the robot will get (and stop) in the goal after the next step. If $3.8 < s \leq 4$ the robot is already in the goal, but it may leave it with its next step. A successful plan must not only make sure that the system achieves a goal state, but that it also remains in a goal state after the execution of the plan.

If the robot had a way to verify goal achievement, we could derive the following guaranteed plan: "**while** (*not in the goal*) **execute** F." This is an example of a generalized plan structure, where the plan ceases to be linear and becomes an algorithm that is parsed and executed by the robot. In order to facilitate decisions during the plan execution, the robot observes the state s and transforms it into a variable $\hat{s}$ that is stored internally or fed to the execution module directly. This transformation is called *perception* and is modelled with the function $\hat{s} = \mathbf{per}(s)$. However, decisions like goal achievement, need to be functions of the actual state of the world s. Therefore, we need to be able to obtain estimates of the actual state s from the perceived state $\hat{s}$. Let us call this mapping *interpretation*: $s = \mathbf{int}(\hat{s})$.

Perceptual Uncertainty: If the perception function is one-to-one, then interpretation is a one-to-one function, too. Each value of s corresponds to a unique value of

$\hat{s}$. Unfortunately, this is never the case in the real world. First, perception is a lossy procedure: we cannot possibly record all parameters of a real system. Secondly, **per** is not even deterministic because of measurement errors; it is a stochastic function. In general, both perception and interpretation are many-to-many mappings. We call this fact *perceptual uncertainty.* We can model perceptual uncertainty, with the introduction of functions **PER** and **INT** that accept probability distributions as arguments, and return probability distributions: $f_{per}(\hat{s}) = \mathbf{PER}(f(s))$ and $f_{int}(s) = \mathbf{INT}(f(\hat{s}))$.

Back to our example, let us assume that: $\mathbf{PER}(u[a,b]) = u[a-0.1, b+0.1]$ and $\mathbf{INT}(u[a,b]) = u[a-0.1, b+0.1]$. The straightforward condition "*not in the goal*" must now be transformed either into a condition on $\hat{s}$, or, equivalently, into a condition on all possible interpretations of $\hat{s}$.

Consider the plan "**while** ($\hat{s} < 2.6$) **execute** F." Figure 2.1 demonstrates the execution of this plan. Dark-shaded areas correspond to all possible actual locations of the robot (derived only by the knowledge of the initial state of the robot prior to the execution of a command, and the accuracy of the controller). White boxes represent all possible actual readings of the robot's sensor.

In the first line of the figure the initial situation is shown: The robot lies somewhere in $[-0.6, 0.4]$ and its sensor may read anything in the range $[-0.7, 0.5]$. Since all possible sensor readings are lower that the cutoff value of 2.6, the robot will necessarily take a step forward. In fact, the robot will have to take two more steps forward before its sensory reading may exceed 2.6.

The situation is depicted in the second line of Figure 2.1. After three steps, s may lie anywhere in $[1.8, 4]$ and $\hat{s}$ anywhere in $[1.7, 4.1]$. If $\hat{s} \geq 2.6$, the execution of the plan will stop. However, if $\hat{s} \in [1.7, 2.6)$, the execution will proceed for at least one more step. If this happens, using the interpretation function and our prior knowledge about s, we know that s must lie in $[1.8, 2.7)$, as shown in the third line of Figure 2.1.

The fourth line of the figure depicts the situation after such a fourth step: The range of values of s is $[2.6, 3.9)$, and the range of values of $\hat{s}$ is $[2.5, 4)$. Again, if $\hat{s} \geq 2.6$ the execution will halt, otherwise it will proceed for a fifth step; s will start in $[2.6, 2.7)$ (line 5 of Figure 2.1), and will end up in $[3.4, 3.9)$. The new $\hat{s}$ will lie in $[3.3, 4)$, so the execution will terminate here (line 6 of the figure). In all cases,

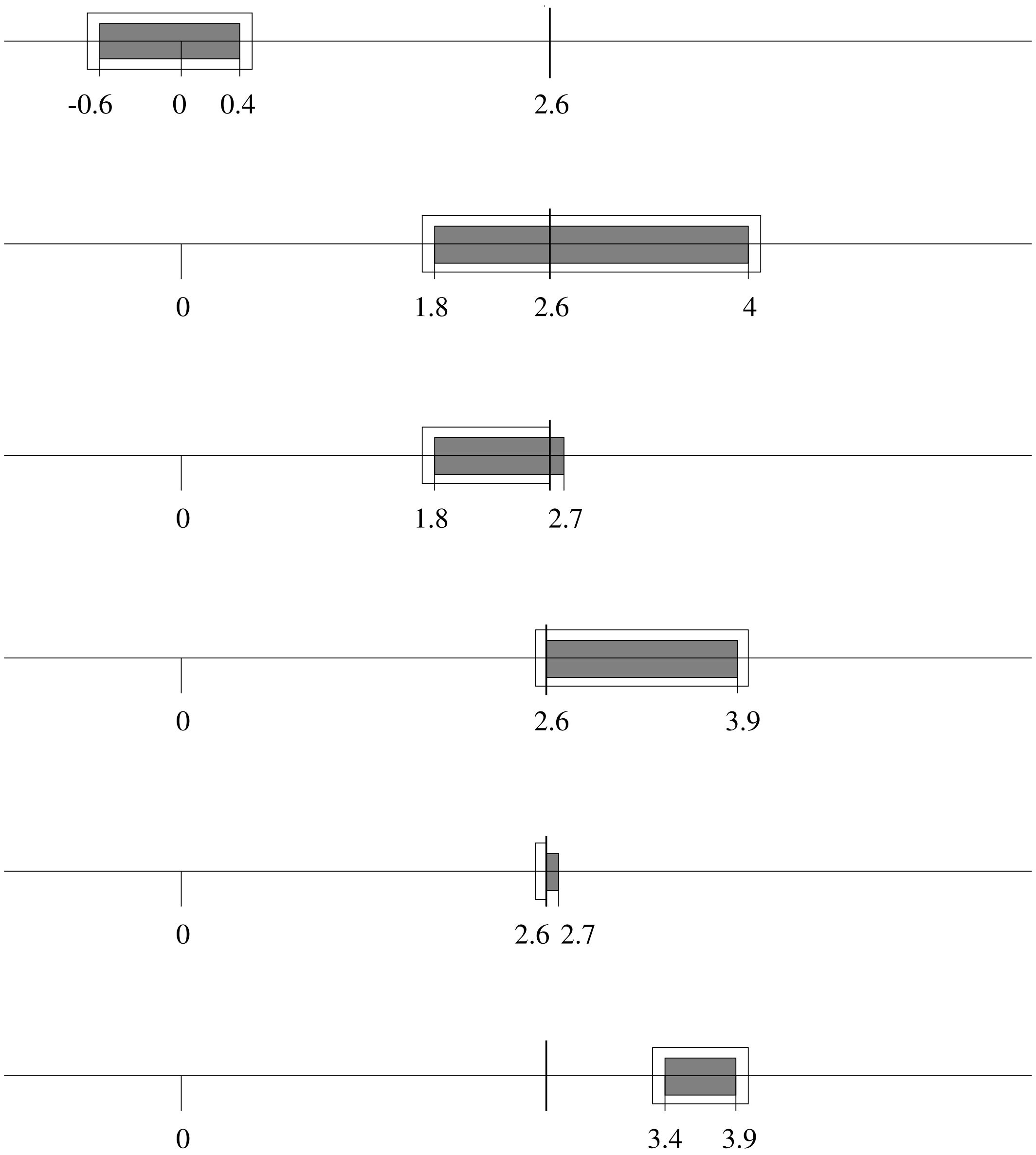


Figure 2.1: Planning with perceptual uncertainty

after the plan terminates, the system will be in a goal state $[2.5, 5]$, i.e., the plan is guaranteed.

Note that it is possible to terminate the execution of the plan after the fourth step, because we know that the actual location of the robot lies in $[2.6, 3.9]$, i.e., necessarily in the goal. However, since our termination condition is based solely on $\hat{s}$, it is not possible to do so. As we will see in the following chapters, it is much harder to derive plans that take into consideration the derived actual state of the robot.

Another interesting observation is that during plan execution the robot will have more accurate knowledge of its actual location s, because, after each step, the interpretation function will narrow down the possible values of s into a 0.2-wide interval. However, during the planning phase we need to take into account *all* possible situations that may result during plan execution, because we are interested in guaranteed plans. This observation can be extended to argue that while the planning module must use appropriate models for fast and guaranteed planning, the execution module may use different (yet compatible), more accurate models, so that the execution of plans becomes more efficient.

2.4 Deriving one-step plans

2.4.1 Algorithmic formulation of plan execution

Planning with uncertainty is quite complicated. In order to come up with a problem that we can solve rigorously, we will make a few simplifying assumptions. As a first step, we will not use probability density functions to represent beliefs about the state of the system. Instead, we will assume that beliefs are limited to the knowledge of a set, representing all values of s currently believed possible. We thus represent a density function $f(s)$ with the set $F = \{s \mid f(s) > 0\}$. According to this representation, $\mathbf{PER}(S)$ represents the set containing all values of $\hat{s}$ that can be observed, if s can take any value in S; $\mathbf{INT}(\hat{S})$ is the set containing all values of s that are compatible with a measurement $\hat{s} \in \hat{S}$; and $\mathbf{ACT}(c, S)$ is the set containing all possible values of s after the execution of command c in a system starting at any state $s \in S$. We also introduce the function $\mathbf{dec}(S, P)$ which returns the command prescribed by plan P when the state of the system can take any value $s \in S$. Now we can use the following algorithm to describe the execution of a plan P in a system which starts at an initial state known to be in a set S_0:

```
$\hat{s} \in \mathbf{PER}(S_0)$ is the perceived initial state of the system;
$\tilde{S} \leftarrow \mathbf{INT}(\{\hat{s}\}) \cap S_0$;
while ($\tilde{S} \not\subseteq \mathbf{G}$) do {
    $c \leftarrow \mathbf{dec}(\tilde{S}, P)$;
    execute $c$;
    $S \leftarrow \mathbf{ACT}(c, \tilde{S})$;
    $\hat{s} \in \mathbf{PER}(S)$ is the perceived new state of the system;
    $\tilde{S} \leftarrow \mathbf{INT}(\{\hat{s}\}) \cap S$;
}
return $success$;
```

2.4.2 The preimage and the kernel

Suppose we are interested in finding a *one-step* plan, i.e., a plan, all the executions of which will traverse the loop of the execution algorithm just once. The question whether a one-step guaranteed plan exists can be formulated as follows:

$$\forall \hat{s} \in \mathbf{PER}(S_0),\ \exists c:\ \mathbf{ACT}(c, \mathbf{INT}(\{\hat{s}\}) \cap S_0) \subseteq \mathbf{G}$$

Essentially, deriving a one-step plan entails associating a command c with all possible values of $\hat{s}$. This number is always finite (since $\hat{s}$ is the output of a physical instrument), but it may be very large. It would be a tedious and time consuming task to have the planner reason explicitly about each of these values separately.

Fortunately, there is a better way. Consider the *preimage* function (**PRE**), which operates inversely to **ACT**: For a given set of system states S and a command c, it returns the *maximal* set of system states S_P such that: $S_P = \mathbf{PRE}(c, S) \Rightarrow \mathbf{ACT}(c, S_P) \subseteq S$. In other words, the preimage of a set S of system states for a command c is another set of system states S_P with the following properties: a) If c is executed with the system starting in any state in S_P, the resulting state is guaranteed to be in S. b) There is no set with the above property that is not a subset of S_P. An interesting property of the preimage, which follows directly from its definition, is that the preimage of a subset of a bigger set is a subset of the preimage of the bigger set for the same command.

If computation of preimages is possible, then the following algorithm can be used to solve the one-step planning problem:

```
$S \leftarrow S_0$;
$\hat{S} \leftarrow \mathbf{PER}(S_0)$;
while ($\hat{S} \neq \emptyset$) do {
    select a new command $c$ that has not been already processed;
    if (all commands have been processed) return failure;
    $P \leftarrow \mathbf{PRE}(c, \mathbf{G})$;
    $S^+ \leftarrow S \cap P$;
    $S^- \leftarrow S \setminus P$;
    $K \leftarrow S^+ \setminus \mathbf{INT}(\mathbf{PER}(S^-))$;
    if ($K \neq \emptyset$) associate $c$ with each value $\hat{s} \in \mathbf{PER}(K)$;
    $S \leftarrow S \setminus K$;
    $\hat{S} \leftarrow \hat{S} \setminus \mathbf{PER}(K)$;
}
return success;
```

This algorithm attempts to cover $\mathbf{PER}(S_0)$ with a collection of sets $\mathbf{PER}(K^i)$, and associate a command c^i with each of these sets. If $\mathbf{PER}(K^i) \cap \mathbf{PER}(K^j) \neq \emptyset$ with $i \neq j$, then the states in this intersection have the following property: The command prescribed by the plan, if the system initially lies in one of these states, may vary in different plan executions.

The set $\mathbf{KER}(S^+, S) \equiv S^+ \setminus \mathbf{INT}(\mathbf{PER}(S \setminus S^+))$ is called the *kernel* of the set S^+, and it has a special significance in reasoning about perception under uncertainty. Suppose we know that the state of the system s lies in a set S, and we wish to derive a condition based on the perceived value of the system state $\hat{s}$ so that, if the condition is true, the state is guaranteed to lie in some prespecified set S^+. This condition is simply $\hat{s} \in \mathbf{PER}(\mathbf{KER}(S^+, S))$. The structure $\mathbf{INT}(\mathbf{PER}(S \setminus S^+))$ represents all the system states that may produce the same perceived value $\hat{s}$ with some system state not in S^+, i.e., all system states that can be *confused* with a state outside S^+. Subtracting these states from our target set S^+ leaves only those states in S^+ that cannot be confused with feasible states outside S^+.

2.4.3 An example

Let us examine how the above ideas apply to a particular example. Consider again the point robot that moves on a straight line. Let its initial position be known to be in $[-0.6, 0.4]$, and let the goal be [2.5,5]. This time, however, the command set that is available to the user is richer: $\{M_i, \mid i = 0, \pm 1, \pm 2, \ldots, \pm m\}$. The action function is defined as follows:

$$\mathbf{ACT}(M_i, [a, b]) = [a + 0.8 \times i,\ b + 1.2 \times i],$$

hence the preimage function is

$$\mathbf{PRE}(M_i, [a, b]) = \begin{cases} \emptyset, & \text{if } 0.4 \times i > (b - a); \\ [a - 0.8 \times i,\ b - 1.2 \times i], & \text{otherwise.} \end{cases}$$

Keeping the same perception and interpretation function as before, the possible initial perceived values are included in $[-0.7, 0.5]$, the perception set. The planning algorithm visits all commands in arbitrary order (say, in increasing order of i), and computes the preimage of the goal for this command and the kernel of the intersection of the preimage with the initial set. The first command with a non-empty kernel is M_3. The preimage of the goal for this command is $[0.1, 1.4]$, so S^+ is $[0.1, 0.4]$, S^- is $[-0.6, 0.1)$, and $\mathbf{INT}(\mathbf{PER}(S^-)) = [-0.8, 0.3)$. Now we can compute the kernel to be $[0.3, 0.4]$ and its perception set to be $[0.2, 0.5]$. Thus, whenever the robot perceives an initial value of $\hat{s}_0$ in $[0.2, 0.5]$, it should execute M_3. We shrink the initial set to $[-0.6, 0.3)$ and the perception set to $[-0.7, 0.2)$.

We move to the next command M_4. The backprojection of the goal for this command is $[-0.7, 0.2]$, S^+ is $[-0.6, 0.2]$ and S^- is $(0.2, 0.3)$. We compute the kernel to be $[-0.6, 0]$, and we associate M_4 with any value of $\hat{s}_0$ in $[-0.7, 0.1]$. The initial set is shrunk to $(0, 0.3)$ and the perception set to $(0.1, 0.2)$.

Unfortunately, no other command produces a backprojection with a non-empty intersection with the initial set, so the planner returns failure. Indeed, no guaranteed plan exists. Imagine that we get an initial reading $\hat{s}_0 \in (0.1, 0.2)$. According to the interpretation function, the actual position of the robot can be less than 0.1 or greater than 0.2, but we have no way to know it. If for this value of $\hat{s}_0$ we command M_3 and the actual position of the robot is less than 0.1, then the resulting actual position of the robot can be less than 2.5, i.e., outside the goal. If, on the other hand, we command M_4 and the actual position of the robot is greater than 0.2, the resulting actual position may be greater than 5, once again outside the goal. Hence, there is no safe command to associate with an initial measurement in the set $(0.1, 0.2)$, a fact that was discovered by our planner, too.

2.4.4 Conditional plans

Let us now consider the same example but with slightly smaller control uncertainty:

$$\mathbf{ACT}(M_i, [a, b]) = [a + 0.85 \times i,\ b + 1.15 \times i],$$

and

$$\mathbf{PRE}(M_i, [a, b]) = \begin{cases} \emptyset, & \text{if } 0.3 \times i > (b - a); \\ [a - 0.85 \times i,\ b - 1.15 \times i], & \text{otherwise.} \end{cases}$$

In this case, the kernel that corresponds to M_3 is found to be $[0.15, 0.4]$, and the kernel that corresponds to M_4 afterwards is $[-0.6, 0.15)$. Their perception sets cover all possible initial readings of $\hat{s}_0$, so we get the following guaranteed plan:

```
**if** ($\hat{s}_0 \in [0.25, 0.5]$) **execute** $M_3$;
**else if** ($\hat{s}_0 \in [-0.7, 0.05]$) **execute** $M_4$;
**else if** ($\hat{s}_0 \in (0.05, 0.25)$) **execute** $M_3$ **or** $M_4$;
```

The above is an example of a *conditional* plan, where the selected command depends on the initial perceived state, as opposed to a *linear* plan, where the choice of command is independent of the initial measurement. Conditional plans are much more powerful than linear plans, because they solve a richer class of planning problems. However, complete planners that return conditional plans are more complicated to build than their linear counterparts. Indeed, if we are seeking only linear plans, we must just find a command for which the preimage of the goal fully includes the initial set.

2.5 Multi-step plans

As we saw in the previous example, a guaranteed one-step plan may not exist. In this case the planner must attempt to find a multi-step plan. Such plans have the advantage of allowing perception actions between the execution of commands, thus diminishing the uncertainty about the state of the system. The planner must reason about what information might become available to the robot at execution time, and try to create a plan that allows the robot to gather as much of this information is necessary in order to reliably achieve the goal.

In the example of the previous section with the higher perceptual uncertainty, we were not able to produce a one-step guaranteed plan when $\hat{s}_0 \in (0.1, 0.2)$. However, the following two-step plan is guaranteed:

```
execute $M_2$
get measurement $\hat{s}$
if ($\hat{s} \in (1.5, 2)$) execute $M_2$
else execute $M_1$
```

To verify that this plan is guaranteed, notice that the possible initial positions of the robot lie in the set $(0, 0.3)$. The first M_2 command transforms this set into $(1.6, 2.5)$, and the measurement $\hat{s}$ can take any value in $(1.5, 2.6)$. According to the plan, if $\hat{s} \in (1.5, 2)$, another M_2 is executed. In this case the possible starting positions of the robot lie in the set $(1.6, 2.1)$, so its possible final positions lie in $(3.2, 4.5)$, i.e., in the goal. Otherwise, $\hat{s}$ is in $[2, 2.6)$, the robot starts from $[1.9, 2.5)$, executes M_1, and ends up in $[2.7, 3.7)$, once again in the goal.

The general problem of whether a guaranteed n-step plan exists can be formulated as follows:

$$\begin{aligned}
&\forall \hat{s}_0 \in \mathbf{PER}(S_0),\ \exists c_0 :\ S_1 = \mathbf{ACT}(c_0, \mathbf{INT}(\{\hat{s}_0\}) \cap S_0),\\
&\forall \hat{s}_1 \in \mathbf{PER}(S_1),\ \exists c_1 :\ S_2 = \mathbf{ACT}(c_1, \mathbf{INT}(\{\hat{s}_1\}) \cap S_1),\\
&\qquad\qquad\qquad\qquad\qquad \ldots\\
&\forall \hat{s}_{n-1} \in \mathbf{PER}(S_{n-1}),\ \exists c_{n-1} :\ \mathbf{ACT}(c_{n-1}, \mathbf{INT}(\{\hat{s}_{n-1}\}) \cap S_{n-1}) \subseteq \mathbf{G}
\end{aligned}$$

Finding a multi-step plan is a much harder problem than its one-step counterpart. The most common technique for finding multi-step plans under uncertainty is *preimage backchaining.* If the preimages of the goal do not cover the initial set in an appropriate way, they are considered as intermediate goals, and the same algorithm is called upon them, and so forth, until the entire initial set has been covered. In the following chapters we will study several versions of this powerful technique.

2.6 Commands with termination condition

In the previous section we have assumed that the robot reassesses its behavior as fast as it can perceive its environment and interpret the sensory readings. This assumption places a tremendous burden on planners that attempt to fully exploit the capability of robots for such adaptive behavior. This often yields very long plans, and thus planning efficiency is seriously affected.

In practice, the action and the perception-interpretation loops do not operate at the same frequency; the action loop is usually much slower. A command consists of a *control statement* **cs** and a *termination condition* **tc**, and prescribes the following behavior for the robot: "**while**($\neg$**tc**) **obey cs**." In general, the termination condition depends on the most recent estimate of the state of the system. The robot begins to alter the state according to **cs**; at the same time it keeps perceiving the current state, updating its estimate about the true state of the system, and evaluating the termination condition continuously. When **tc** evaluates to *true*, the robot stops obeying **cs** and seeks another command from the plan. Putting the above description into an algorithm, we have:

```
$\hat{s} \in \mathbf{PER}(S_0)$ is the perceived initial state of the system;
$\tilde{S} \leftarrow \mathbf{INT}(\{\hat{s}\}) \cap S_0$;
while ($\tilde{S} \not\subseteq \mathbf{G}$) do {
    $(\mathbf{cs}, \mathbf{tc}) \leftarrow \mathbf{dec}(\tilde{S}, P)$;
    adjust the robot's behavior to conform with cs;
    while ($\neg\mathbf{tc}(\tilde{S})$) do {
        $S \leftarrow \mathbf{ACT}(\mathbf{cs}, \tilde{S})$;
        $\hat{s} \in \mathbf{PER}(S)$ is the perceived new state of the system;
        $\tilde{S} \leftarrow \mathbf{INT}(\{\hat{s}\}) \cap S$;
    }
}
return success;
```

However general the above formulation may seem, real systems behave in more convoluted ways. In fact, in real systems the outcome of the **ACT** function may depend on itself, yielding the recursive equation $S \leftarrow \mathbf{ACT}(\mathbf{cs}, S, \tilde{S})$. The simplest such situation occurs when we consider time t (a member of s in the above). Usually, the outcome of **ACT** depends on the elapsed time, i.e., on the value of one member of s. Many workarounds can be proposed (like splitting s into vectors with different constraints and/or properties), but for the purposes of this discussion we will ignore time (or more precisely its effects on **ACT**), and we will retain the notation $\mathbf{ACT}(\mathbf{cs}, \tilde{S})$ to represent all states achievable by the system if **cs** is executed continuously with no termination condition.

In the above algorithm the execution module keeps updating its estimate about the current state of the system with each new value of $\hat{s}$ it obtains. Let h, the *sensing history*, denote the series of all observed values of $\hat{s}$, starting with the one that led to the selection of the particular command $(\mathbf{cs}, \mathbf{tc})$, and up to the most recent one. We can always obtain the most current estimate about the state of the system using only S_0 and h. Therefore, the termination condition depends only on h and S_0. This most general form of the termination condition (that makes use of all available information) is called a termination condition with *history* and *state*.

A plan which can be described with the above formulation, makes use of all available information. This fact contributes to the complexity of the planning problem significantly. In what follows, we will study a restricted class of termination conditions, namely, those that do not depend on the sensing history. Therefore, we will assume that the robot does not update its estimate about the state with each sensing action. The execution algorithm of such a system looks like follows:

```
$\hat{s} \in \mathbf{PER}(S_0)$ is the perceived initial state of the system;
$\tilde{S} \leftarrow \mathbf{INT}(\{\hat{s}\}) \cap S_0$;
while $(\tilde{S} \not\subseteq \mathbf{G})$ do {
    $(\mathbf{cs}, \mathbf{tc}) \leftarrow \mathbf{dec}(\tilde{S}, P)$;
    adjust the robot's behavior to conform with $\mathbf{cs}$;
    $S \leftarrow \mathbf{ACT}(\mathbf{cs}, \tilde{S})$;
    while $(\neg\mathbf{tc}(\tilde{S}))$ do {
        $\hat{s} \in \mathbf{PER}(S)$ is the perceived new state of the system;
        $\tilde{S} \leftarrow \mathbf{INT}(\{\hat{s}\}) \cap S$;
    }
}
return success;
```

Since the set S remains constant throughout the evaluation of **tc**, we can essentially treat **tc** as a function of $\hat{S}$. Furthermore, **tc** is a boolean function, so we can represent it with two sets $\hat{S}_{\mathbf{tc}}$ and $\hat{S}_{\neg\mathbf{tc}}$, where $\hat{S}_{\mathbf{tc}} \cup \hat{S}_{\neg\mathbf{tc}} = \hat{\Omega}$. The first contains all values of $\hat{s}$ that cause **tc** to evaluate to *true*, whereas the second contains all values that cause **tc** to become *false*. Their union $\hat{\Omega}$ is the set of all possible measurements.

Consider again a one-step plan. Let us assume that for a particular execution of the plan the command $(\mathbf{cs}, \mathbf{tc})$ has been chosen. In order to guarantee that the execution will be successful, we have to make sure that at some point $\mathbf{tc}(\hat{S})$ will evaluate to *true*, and that the first time that this happens the system will be in a goal state. The function **ACT** is not adequate to facilitate reasoning like the above. Because **cs** is executed continuously, we need a function that can return possible *trajectories* of s instead of possible values of it. We call this function the *forward* *projection* (**FP**).

Definition 2.1 (Forward Projection) *The forward projection* $\mathbf{FP}(\mathbf{cs}, S)$ *of a set of system states* S *for control statement* **cs** *is the set of possible trajectories of the system state* s*, that occur when the system, initially in some state in* S*, changes in response to the continuous execution of* **cs**.

Note that the set $\mathbf{ACT}(\mathbf{cs}, S)$ consists of all the points of the trajectories included in $\mathbf{FP}(\mathbf{cs}, S)$. Similarly, instead of the function **PRE** we use the *backprojection* function **BP**, defined as follows:

Definition 2.2 (Backprojection) *The backprojection* $\mathbf{BP}(\mathbf{cs}, S)$ *of a set of system states* S *consists of all system states, such that if a system initially being in one of these states starts changing by obeying* **cs***, all possible trajectories of its state will pass through* S.

We can now use these definitions to derive a one-step planning algorithm for such a system. For each possible observable initial state of the system we need to compute a pair $(\mathbf{cs}, \mathbf{tc})$, such that all possible trajectories of the system state that are compatible with **cs** pass through the goal, i.e., we have to make sure that the goal is *reachable*. Reachability, though, is not a sufficient condition for the correctness of a plan. We also need to make sure that the system will stop following **cs** while it is still in the goal, i.e., that along each possible trajectory there exists a point for which **tc** will evaluate to *true*, and that the first time that this happens, this point will lie in the goal. This is the problem of goal *recognizability*.

The set $\mathbf{INT}(\hat{S}_{\mathbf{tc}})$ contains all the states that may cause **tc** to evaluate to *true*. Also, the set $\mathbf{K}_{\mathbf{tc}} = \mathbf{INT}(\hat{S}_{\mathbf{tc}}) \setminus \mathbf{INT}(\hat{S}_{\neg\mathbf{tc}})$ contains all the states that always cause **tc** to evaluate to *true*. As it can be verified, $\mathbf{K}_{\mathbf{tc}} = \mathbf{KER}(\mathbf{INT}(\hat{S}_{\mathbf{tc}}), \Omega)$, where Ω is the set of all possible states of s, hence $\mathbf{K}_{\mathbf{tc}}$ is called the *kernel* of **tc**.

Definition 2.3 (Kernel of a termination condition) *The kernel of a termination condition consists of all system states which* always *give rise to perceived states that cause the termination condition to evaluate to true.*

Clearly, all possible trajectories of a one-step guaranteed plan must lead the system into the intersection of the goal with the kernel of the termination condition, without first passing through a point that belongs to $\mathbf{INT}(\hat{S}_{\mathbf{tc}})$ but not to the goal. Thus, a planner should go over all possible combinations of **cs** and **tc**, and for each such pair compute the backprojection of $\mathbf{K_{tc}} \cap \mathbf{G}$ treating the set $\mathbf{INT}(\hat{S}_{\mathbf{tc}}) \setminus \mathbf{G}$ as an artificial obstacle to be avoided. If several of these backprojections can cover the initial set S_0 in the way described in Section 2.4, then we got a one-step guaranteed plan.

2.7 Selecting a termination condition

An efficient planner cannot afford to examine all possible termination conditions in order to determine the best one. Instead, planners must use heuristic algorithms to select a "nice" termination condition that depends only on the selected control statement.

The properties of a "nice" termination condition can be derived by our desire to obtain as big a backprojection as possible. The size of the intersection of its kernel with the goal is not always a reliable heuristic. What is of real importance is the size of the boundary of this intersection that is not blocked by obstacles (including the artificial obstacles induced by the termination condition). For example, if we use a **tc** that is always true, the intersection of its kernel with the goal is the entire goal, but it is totally surrounded by the artificial obstacle, which includes all non-goal system states. Thus, there exists no control statement that can lead the system into the goal from any state outside the goal. The backprojection for this termination condition is empty, although the size of the intersection of its kernel with the goal is the maximum possible. In other words, the power of the termination condition depends on its ability to discriminate between points in the goal and points outside it. A typical heuristic creates termination conditions which have the maximal kernel without inducing any artificial obstacles.

Let us now, for one final time, visit our example, to demonstrate the above ideas. We use again the command set (F, B), only this time we assume that these commands are executed continuously (i.e., they are control statements) until some termination condition becomes true. We assume that we know nothing about the speed of motion of the robot, thus the function **ACT** is defined as follows:

$$\mathbf{ACT}(c,[a,b]) = \begin{cases} [a,+\infty), & \text{if } c = F; \\ (-\infty,b], & \text{if } c = B. \end{cases}$$

Since the robot moves on a line, it will eventually get into any non-empty set that lies in the direction of its motion, so the backprojection function has the following description:

$$\mathbf{BP}(c,[a,b]) = \begin{cases} (-\infty,b], & \text{if } c = F; \\ [a,+\infty), & \text{if } c = B. \end{cases}$$

Consider the same one-step planning problem, with $S_0 = [-0.6, 0.4]$ and $\mathbf{G} = [2.5, 5]$. How can we select a "nice" termination condition? In this case where the robot moves on a line, the only consideration is not to create an artificial obstacle between the initial set and the goal. Thus "$\hat{s} \geq 2.6$" is a good termination condition: It yields $\hat{S}_{\mathbf{tc}} = [2.6, +\infty)$, $\hat{S}_{\neg\mathbf{tc}} = (-\infty, 2.6)$, $\mathbf{INT}(\hat{S}_{\mathbf{tc}}) = [2.5, +\infty)$, $\mathbf{INT}(\hat{S}_{\neg\mathbf{tc}}) = (-\infty, 2.7)$, and $K_{tc} = [2.7, +\infty)$. The intersection of the kernel with the goal is $[2.7, 5]$, and the artificial obstacle $(5, +\infty)$ is safely behind the goal, allowing the preimage of $[2.7, 5]$ for control statement F to fully enclose the initial set. In this case the plan "**while** ($\hat{s} < 2.6$) **obey** F" is guaranteed.

If, however, we use the termination condition "$\hat{s} \geq 2.5$," it will create the artificial obstacle $[2.4, 2.5) \cup (5, +\infty)$ which fully encloses the goal: The robot cannot enter the goal without getting in danger to stop before it reaches it.

Chapter 3

Robot Motion Planning with Uncertainty

The difference between a robot and a computer is the ability of the former to exert forces upon the physical world, thus altering its configuration. Therefore, any task that is performed by a robot (but cannot be performed by a computer) must involve physical motion of the robot or components of its environment. Motion planning is the foundation upon which most robotic reasoning systems must be built. Unfortunately, motion planning is a very hard problem, and it is not surprising that a lot of effort has been directed by researchers towards this area. A comprehensive analysis of the various motion planning techniques can be found in [53].

Of all the parameters that describe the system of the robot and its environment, we separate those that can be changed, and we call them the *configuration* q. The number of dimensions of the configuration is referred to as the dimensionality of a motion planning problem. The complexity of motion planning depends to a great extent on the dimensionality of the problem.

Since no more than one physical object can occupy the same physical space, not all values of q are possible. We describe this fact with a set of constraints $\{\mathcal{B}_1(q), \ldots, \mathcal{B}_{n_b}(q)\}$, each corresponding to the existence of an *obstacle* in the workspace. If a particular value of q results in a collision (i.e., violates one of the constraints) it is called *infeasible*; otherwise it is called *feasible*. In Chapter 2 we implicitly assume that the existence of obstacles is incorporated into the function **act**, which never returns an infeasible system configuration. An alternative approach is to define **act** in the absence of obstacles, and reason about them explicitly.

In this chapter we make the ideas presented in the previous chapter concrete, by presenting a practical algorithm for planning the motion of a robot under control and sensing uncertainty. The algorithm uses the method of "preimage backchaining." We consider three sensing schemes (sticking, position sensing, force sensing) and their combination. Several planning problems, solved with the implemented algorithm, are shown. Finally, we discuss the notion of maximal preimages and show that they do not exist.

3.1 Background

3.1.1 The path-planning problem

Research in motion planning began in the mid-seventies. Some early results can be found in [92, 66, 65]. It was soon understood that the pure topological and geometrical aspects of motion planning constituted an interesting problem of its own, and a lot of effort was directed towards its solution. This problem, the *path-planning problem*, is the search for a path of feasible configurations (a collision-free path) starting at the initial state and ending at a goal state. The robot is assumed to be able to follow the path without error. The archetype of such problems is the so called *piano-movers problem*. When the dimensionality is not fixed, the path-planning problem has been shown to be PSPACE-hard [85]. Other theoretical results regarding lower and upper bounds of multiple variants of the problem can be found in [88]. When the dimensionality is fixed, algorithms have been proposed, which are polynomial in the number of algebraic surfaces bounding the workspace and the obstacles and their maximal degree [87, 13].

These results are of theoretical interest, but have little practical use in systems with high dimensionality, even when uncertainty is not an issue. Several other planning techniques have been proposed in an attempt to find more practical algorithms [11, 45, 57, 68]. Most of them trade *completeness* for *efficiency*, i.e., they search a smaller space with the hope that it contains at least one solution. The *artificial potential field* method, introduced by Khatib [50], has been widely adopted and has resulted in many practical motion planning algorithms, like the one described in [38]. The idea is that obstacles exert repulsive forces to the robot, whereas the goal attracts it. These forces generate a potential field in the workspace having a global minimum at the goal. The robot moves following the gradient of the potential field until it reaches a minimum; if the minimum is local, the robot tries to escape, otherwise the goal has been achieved. Detecting and escaping local minima is the major problem of potential-field-based methods. Recently, the introduction of randomized and parallel algorithms has resulted in very effective techniques to address these problems. Barraquand and Latombe [5] described such a planner that is able to solve complicated motion planning problems with more than 10 degrees of freedom in a few seconds.

3.1.2 Dealing with uncertainty

Few of the algorithms described above can be used independently in real systems. The reason is that they do not take into account uncertainty. Reasoning about uncertainty complicates the motion-planning problem significantly. Moreover, since we cannot anticipate anything that may happen, we have to define precisely the extent to which uncertainty will be dealt with during the planning phase. This issue has been the subject of debate among roboticists and AI researchers, and has led to the formation of two major schools of thought.

The *reactionists* believe that planning should not be concerned with uncertainty. If unexpected situations arise during the execution of a plan, the robot should attempt to react to them following some prespecified *behavior*. If the execution of the plan cannot continue, the planner is asked to provide a new plan. The arguments of this school are (a) that it is not worth to spent too much planning time preplanning about low probability events, and (b) that even if we can anticipate some uncertainty, there are so many unexpected events, that the plan will fail in one way or another; if the robot must deal with such failures, it will also be able to deal with failures that could have been anticipated. Several systems using the above "reactive planning" method have been proposed and implemented with various degrees of success [10, 39, 1, 73]. The major disadvantage of this method is the inability to model the behavior of the robot in a meaningful way. Success or failure cannot be explained, so results cannot be generalized or transferred to a different system. Furthermore, uncertainty exists both at planning time and at execution time. One example of execution time uncertainty is perceptual errors that may prevent pure reactive schemes from completely eliminating uncertainty by reading sensory inputs. Perceptual uncertainty may also hinder the robot to correctly identify its own state, and the state of the workspace. False state identification may trigger inappropriate robot behavior. It is necessary for the robot to reason in advance about the knowledge that may become available at execution time, and make sure that this knowledge will be sufficient to lead it towards the goal or recognize failure.

For these reasons, the other school proposes that a planner should anticipate "as much uncertainty as possible." Typically, the term "as much uncertainty as possible" means that the considered stochastic parameters have distributions with relatively small variance (i.e., we do have some knowledge about these parameters), and that an algorithm that solves the planning problem exists. Unfortunately, most planning-with-uncertainty algorithms that have been proposed up to now require exponential running time. Our view is that it is only worthwhile to reason about uncertainty if the planning algorithm is fast enough to be practical. Moreover, we believe that reactive behaviors (like unexpected obstacle avoidance) should be an integral part of a robust robotic system, because there is no way that a planner can anticipate *everything* that may happen *all* the time.

3.1.3 Planning with uncertainty

Four major approaches to planning with uncertainty have been proposed so far. The first, called *skeleton refinement* was introduced independently by Lozano-Pérez [66] and Taylor [92]. It consists of (a) retrieving a plan skeleton appropriate to the task at hand and taking that as an initial plan, and (b) iteratively modifying the skeleton by inserting complements (typically sensor readings). Complements are decided after the correctness of the skeleton is checked, either by propagating uncertainty through the steps of the plan skeleton [92] or by simulating several possible executions [66]. Subsequent contributions to the approach have been brought by Brooks [9], who developed a symbolic computation technique for propagating uncertainty forward and backward through plan skeletons, and by Pertin-Troccaz and Puget [81], who proposed techniques for verifying the correctness of a plan and amending incorrect plans.

The second approach to motion planning with uncertainty was proposed by Dufay and Latombe [29], and is known as the *inductive learning* approach. It consists of assembling input partial strategies into a global one. First, during a training phase, the system uses the partial strategies to make on-line decisions and execute several instances of the task at hand. Then, during an induction phase, the system combines the execution traces generated during the training phase, and generalizes them into a global strategy. In fact, the training phase and the induction phase are interweaved. The generation of a strategy for the task ends when new executions do not modify the current strategy. A system based on these principles has been implemented, and experimented successfully on several part mating tasks.

A third method, called *iterative removal of contacts* creates plans using compliant motions on obstacle faces. Compliant motions have the advantage that they completely eliminate uncertainty along the normal to the contact surface. Several variants of this method have been proposed [58, 44, 20].

The fourth approach is called *preimage backchaining* and it will be presented in detail in the following.

3.1.4 Preimage backchaining

The first three approaches essentially operate in two phases: first, a motion plan is generated assuming no uncertainty, and then it is transformed to deal with uncertainty. Instead, preimage backchaining takes uncertainty into account throughout the whole planning process. In principle, it can tackle problems where uncertainty shapes the structure of a plan to the extent that the plan cannot be generated by transforming an initial one produced under the no-uncertainty assumption. This method was first introduced by Lozano-Pérez, Mason and Taylor [69] and has since been accepted as the primary method for the creation of guaranteed plans despite the presence of uncertainty.

The *preimage* of a goal region, for a given motion command $\mathbf{M}$, is the set of all robot configurations such that if the robot starts executing the command from any one of these configurations, it is guaranteed to reach the goal and stop in it. Preimage backchaining consists of constructing a sequence of motion commands $\mathbf{M}_i$ $(i = 1, \ldots, n)$, such that, if $\mathcal{P}_n$ is the preimage of the goal for $\mathbf{M}_n$, $\mathcal{P}_{n-1}$ the preimage of $\mathcal{P}_n$ for $\mathbf{M}_{n-1}$, and so on, then $\mathcal{P}_1$ contains all possible initial configurations of the robot.

One source of difficulty in computing preimages is the interaction between goal reachability and goal recognizability. The robot must both reach the goal (despite initial position and control uncertainty) and stop in the goal (despite sensing uncertainty). Goal recognition often depends on the region from where the command is executed. This region, which is precisely the preimage of the goal for that command, also depends on the way the goal is recognized. This recursive dependence was noted in [69]. Despite this difficulty, Canny [14] described a complete planner with very few restrictive assumptions. This planner generates a plan consisting of r motion commands by reducing the input problem to deciding the satisfiability of a semi-algebraic formula with $2r$ alternating existential and universal quantifiers. Such a decision takes double exponential time in r. Even worse, the smallest r for which a plan may exist grows with the complexity of the environment. Actually, various forms of the above motion planning problem have been proven intrinsically hard [15, 77, 14].

At the expense of completeness, Erdmann [32] suggested that goal reachability and recognizability be treated separately by identifying a subset of the goal, called a *kernel*, such that when this subset is attained, goal achievement can be recognized independently of the way it has been achieved. He defined the backprojection of a region $\mathcal{T}$, for a motion command $\mathbf{M}$, as the set of all points such that, if the robot executes $\mathbf{M}$ starting at any one of these points, it is guaranteed to reach $\mathcal{T}$. He proposed an $O(n \log n)$ algorithm to compute backprojections in the plane when the obstacles are polygons bounded by n edges. An implemented planner based on this approach is described in [55].

3.2 The configuration space

We have previously defined the *configuration* q of a robot and its environment as the values of the changeable parameters of the system. Let us now narrow this definition for pure robot motion planning problems, i.e., for problems where the robot is the only movable object in a workspace $\mathcal{W}$, populated by stationary obstacles $\mathcal{B}_i$, $i \in [1, n_b]$. A *configuration* of the robot is a specification of the position of every point of the robot with respect to a coordinate system embedded in $\mathcal{W}$ [2]. The *configuration space* denoted by $\mathcal{C}$, is the set of all possible configurations.

Each obstacle $\mathcal{B}_i$ is mapped to the subset $\mathcal{CB}_i$ of configurations for which the robot collides with $\mathcal{B}_i$:

$$\mathcal{CB}_i = \{q \in \mathcal{C} \mid \mathcal{R}(q) \cap \mathcal{B}_i \neq \emptyset\},$$

where $\mathcal{R}(q)$ denotes the subset of $\mathcal{W}$ occupied by the robot at configuration q. The region $\mathcal{CB}_i$ is called a *C-obstacle*.

In general, $\mathcal{C}$ is a non-Euclidean manifold. For instance, if the robot is a rigid planar object moving freely in $\mathcal{W} = \Re^2$, then $\mathcal{C} = \Re^2 \times S^1$, where S^1 denotes the unit circle. If the robot is a rigid three-dimensional object moving freely in $\mathcal{W} = \Re^3$, $\mathcal{C} = \Re^3 \times SO(3)$, where $SO(3)$ denotes the Special Orthogonal Group of orthonormal matrices with determinant $+1$ [2].

For the rest of the chapter, we assume that the robot is a two-dimensional polygonal object that can only translate in the plane, e.g., an omnidirectional mobile robot that cannot rotate. A configuration is represented by $q = (x, y) \in \Re^2$, where x and y are the coordinates of a specific point of the robot (the *reference point*) with respect to the coordinate system embedded in $\mathcal{W}$. Hence, both $\mathcal{W}$ and $\mathcal{C}$ are copies of $\Re^2$. Both the robot and the obstacles are modeled as polygonal regions, the robot as a simple polygon, and each obstacle as a region whose boundary is a simple polygonal curve

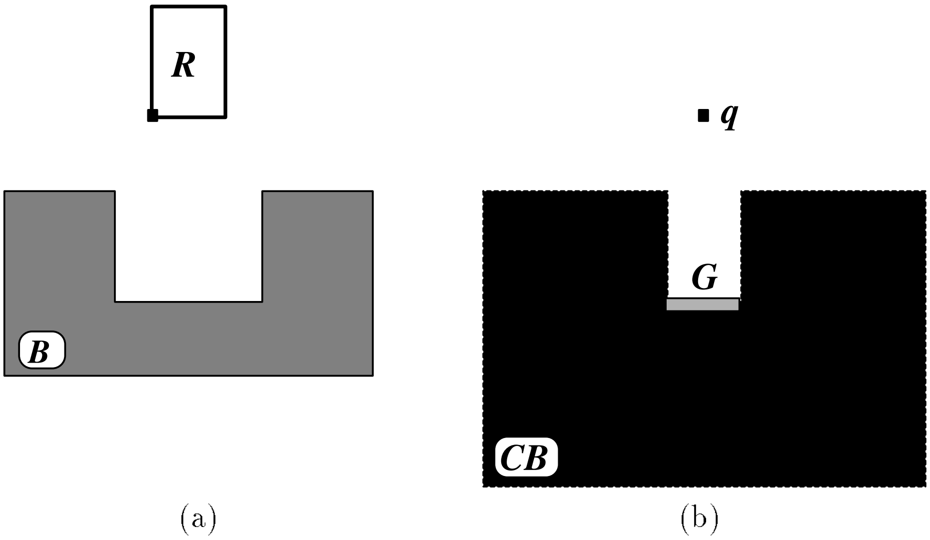


Figure 3.1: Peg-Into-Hole Task

that may, or may not be a closed-loop curve (allowing for non-bounded obstacles). Under these assumptions, each C-obstacle $\mathcal{CB}_i$ is also a polygonal region.

Figure 3.1 illustrates the above concepts. A simple setting in the workspace is depicted in Figure 3.1(a). The robot $\mathcal{R}$ is a rectangle and there is a single polygonal obstacle $\mathcal{B}$ with a rectangular depression. The goal of the task is to insert $\mathcal{R}$ in $\mathcal{B}$'s depression ("peg-into-hole" task). Figure 3.1(b) shows the mapping of this setting in $\mathcal{R}$'s configuration space. The robot $\mathcal{R}$ maps to the point denoted by q. The obstacle $\mathcal{B}$ maps to the C-obstacle $\mathcal{CB}$. The width of the rectangular depression in $\mathcal{CB}$ is equal to the difference between the width of the depression in $\mathcal{B}$ and the width of $\mathcal{R}$. The goal of the peg-into-hole task is to move q (the configuration of $\mathcal{R}$) to any location in the edge $\mathcal{G}$ at the bottom of $\mathcal{CB}$'s depression.

The union of the obstacles, $\bigcup_{i=1}^{n_b} \mathcal{B}_i$, is called the *obstacle region* and is denoted by $\mathcal{B}$. The union of the C-obstacles, $\bigcup_{i=1}^{n_b} \mathcal{CB}_i$, is called the *C-obstacle region* and is denoted by $\mathcal{CB}$. The complement of the C-obstacle region in $\mathcal{C}$ is called the *free space* and is denoted by $\mathcal{C}_{free}$. The subset of configurations q where $\mathcal{R}(q)$ intersects with the obstacle region without overlapping its interior is called the *contact space* and is denoted by $\mathcal{C}_{contact}$. The union of the free space $\mathcal{C}_{free}$ and the contact space $\mathcal{C}_{contact}$ is called the *feasible space* and is denoted by $\mathcal{C}_{feasible}$. A *feasible path* between two configurations q_1 and q_2 in $\mathcal{C}_{feasible}$ is a continuous map $\tau : [0,1] \rightarrow \mathcal{C}_{feasible}$ such that $\tau(0) = q_1$ and $\tau(1) = q_2$.

The set $\mathcal{C}_{free}$ is an open subset of $\mathcal{C}$, whose boundary $\partial\mathcal{C}_{free}$ is a set of polygonal curves. $\mathcal{C}_{contact}$ is also made of polygonal curves. It can be shown that $\partial\mathcal{C}_{free} \subseteq \mathcal{C}_{contact}$ and $cl(\mathcal{C}_{free}) \subseteq \mathcal{C}_{feasible}$ [48]. In Figure 3.1, the strict inclusion of $\partial\mathcal{C}_{free}$ in $\mathcal{C}_{contact}$ occurs when $\mathcal{R}$'s width is exactly equal to the width of $\mathcal{B}$'s depression. Then the depression in $\mathcal{CB}$ degenerates to a line segment contained in $\mathcal{C}_{contact}$, but not in $\partial\mathcal{C}_{free}$. In order to avoid such pathological cases, we assume that each maximally connected subset of the C-obstacle region $\mathcal{CB}$ is homeomorphic[1] to a closed disc or a closed half-space, i.e., is a manifold with boundary. This assumption entails that $\mathcal{C}_{contact} = \partial\mathcal{C}_{free}$ and $\mathcal{C}_{feasible} = cl(\mathcal{C}_{free})$; hence, a contact between $\mathcal{R}$ and the obstacle region can be broken by an arbitrarily small displacement of $\mathcal{R}$. It also implies that every vertex in the contact space $\mathcal{C}_{contact}$ is the extremity of only two edges; thus, $\mathcal{C}_{contact}$ consists of disjoint simple polygonal lines. An edge (resp. a vertex) in $\mathcal{C}_{contact}$ is called a *contact edge* (resp. a *contact vertex*). The *outgoing normal* of a contact edge E is the unit vector $\vec{\nu}(E)$ normal to E pointing toward $\mathcal{C}_{free}$. The vector $-\vec{\nu}(E)$ is the *ingoing normal* of E.

Let $n_{\mathcal{R}}$ be the number of edges of $\mathcal{R}$, and $n_{\mathcal{B}}$ be the number of edges of the obstacle region. $\mathcal{C}_{contact}$ contains $O(n_{\mathcal{R}}^2 n_{\mathcal{B}}^2)$ edges which can be computed in $O(n_{\mathcal{R}}^2 n_{\mathcal{B}}^2 \log n_{\mathcal{R}} n_{\mathcal{B}})$ time [89].

3.3 Control, sensing and uncertainty

3.3.1 Structure of a motion plan

According to the model of Chapter 2, the execution of a motion plan occurs in *steps*. During each step the execution module selects a *control statement* **cs** which prescribes a particular behavior to be followed by the robot, and a *termination condition* **tc** which is a boolean function of the sensory readings. (Time is considered to be one of the possible sensory readings.) The pair $(\mathbf{cs}, \mathbf{tc})$ constitutes a *motion command.*

---

[1] Two topological sets E and F are *homeomorphic* iff there exists a bijection $f : E \rightarrow F$ such that both f and f^{-1} are continuous.

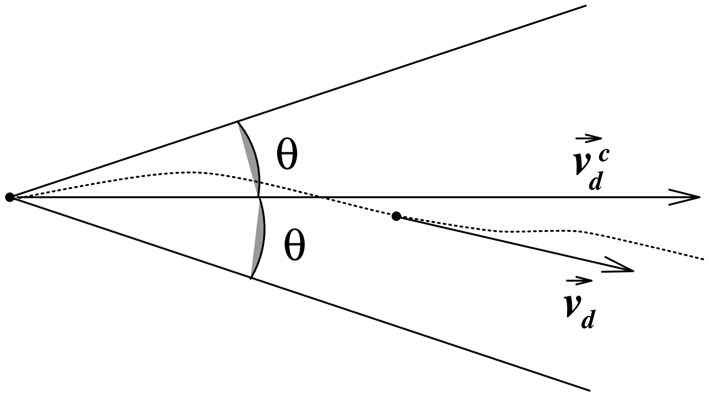


Figure 3.2: The control uncertainty cone

The robot moves obeying **cs**, until **tc** evaluates to true, at which point it stops instantaneously.

The way in which the execution module decides what motion command to execute next depends on the structure of the motion plan. *Linear plans* specify a sequence of motion commands that are to be executed one after the other. *Conditional plans* specify several possible motion commands at each execution step. The robot selects one of them to execute next, based on its beliefs about its own state and the state of the world. These beliefs depend mainly on sensory readings, but may also depend on sensing history, and other prior information. Finally, *randomized plans* use a stochastic function in order to select a motion command at each step. In this chapter we consider only linear plans.

3.3.2 Control

Several schemes have been proposed to control the motion of a robot. Here, we assume that the only control parameter is the *commanded velocity* of motion $\vec{\mathbf{v}}_d^c$. The robot is commanded to move along $\vec{\mathbf{v}}_d^c$, but it is not always possible to do so because of control uncertainty. At any instant during the motion in free space, the *actual velocity* of the robot is a vector $\vec{\mathbf{v}}_d$, such that the magnitude of the angle between $\vec{\mathbf{v}}_d^c$ and $\vec{\mathbf{v}}_d$ is less than a fixed angle $\theta < \pi/2$. In other words, $\vec{\mathbf{v}}_d$ lies in a cone of angle 2θ whose axis points along the d direction. This cone is called the *control uncertainty*

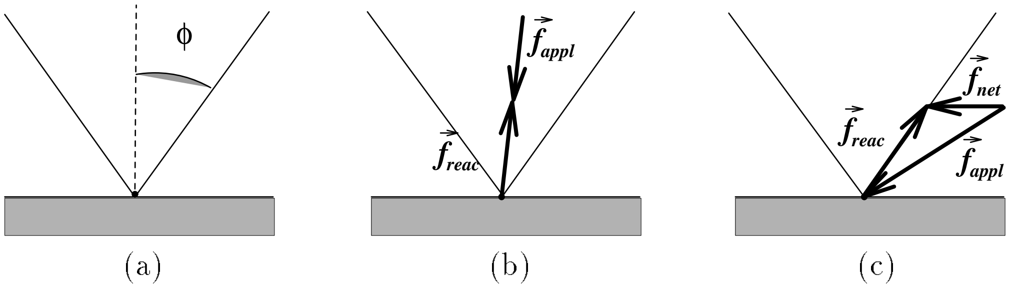


Figure 3.3: Reaction Force

cone. The modulus of the velocity of the robot is unknown to the planner, but is assumed to have some reasonable (positive) value. For simplification, we assume that there is no uncertainty in the velocity modulus, i.e., $\|\vec{\mathbf{v}}_d\| = \|\vec{\mathbf{v}}_d^c\|$. This assumption has no significant impact on the methods described below and can be easily relaxed. For further simplification (and without loss of generality), we assume that $\vec{\mathbf{v}}_d^c$ (hence, $\vec{\mathbf{v}}_d$) is a unit vector, so that the direction d suffices to characterize the vector $\vec{\mathbf{v}}_d^c$ (and the range of vectors $\vec{\mathbf{v}}_d$ compatible with $\vec{\mathbf{v}}_d^c$). During motion in free space, $\vec{\mathbf{v}}_d$ may vary arbitrarily between the two extreme orientations determined by $\vec{\mathbf{v}}_d^c$ and θ. Thus, if the robot is in the free space, it moves along a trajectory whose tangent at any configuration is contained in the control uncertainty cone anchored at this configuration (see Figure 3.2).

The behavior of the robot when it comes in contact with obstacles depends on the particular control scheme we use. Sometimes, contact with obstacles is not allowed. In this case, $\mathcal{C}_{feasible} = \mathcal{C}_{free}$. In most cases, though, compliant motions are allowed, because they eliminate the positional uncertainty along the normal to the contact surface and allow the robot to cover long distances with no uncertainty penalty along one axis.

The control scheme described above can be extended to deal with compliant motion [71, 83, 96]. It is called the *generalized damper compliance model*.

According to this model, the robot exerts a force on its environment that is proportional to the actual commanded velocity $\vec{\mathbf{v}}_d$, say $\vec{\mathbf{f}}_{appl} = B\vec{\mathbf{v}}_d$. When the robot is in free space, no reaction force is applied to the robot, and the force exerted by the robot is entirely used to create motion. When the robot is in contact space and pushes on an obstacle, the obstacle generates a reaction force. If the vector $-\vec{\mathbf{v}}_d$ lies in the friction cone at the contact configuration, the reaction force is $-B\vec{\mathbf{v}}_d$, i.e., it completely cancels the force exerted by the robot, and there is no motion (Figure 3.3(b)). If the vector $-\vec{\mathbf{v}}_d$ lies outside the friction cone, the reaction force only partially cancels the force exerted by the robot and the net resulting force makes the robot's configuration slide in contact space (Figure 3.3(c)).

Let q be the actual configuration of the robot at some instant t. We denote by $\vec{\mathbf{f}}_{reac}(q, \vec{\mathbf{v}}_d)$ the vector in configuration space representing the reaction force applied at the robot's configuration when the actual commanded velocity is $\vec{\mathbf{v}}_d$. $\vec{\mathbf{f}}_{reac}(q, \vec{\mathbf{v}}_d)$ is exhaustively defined as follows:

- if $q \in \mathcal{C}_{free}$, then $\vec{\mathbf{f}}_{reac}(q, \vec{\mathbf{v}}_d) = 0$;
- if $q \in \mathcal{C}_{contact}$, q is in the interior of a contact edge E, and $\vec{\mathbf{v}}_d$ projects positively on the outgoing normal of E, then $\vec{\mathbf{f}}_{reac}(q, \vec{\mathbf{v}}_d) = 0$;
- if $q \in \mathcal{C}_{contact}$, q is in the interior of a contact edge E, and $-\vec{\mathbf{v}}_d$ lies in the friction cone at the contact point, then $\vec{\mathbf{f}}_{reac}(q, \vec{\mathbf{v}}_d) = -B\vec{\mathbf{v}}_d$ (see Figure 3.3(b));
- if $q \in \mathcal{C}_{contact}$, q is in the interior of a contact edge E, and $-\vec{\mathbf{v}}_d$ projects positively on the outgoing normal of E but lies outside the friction cone at the contact configuration, then $\vec{\mathbf{f}}_{reac}(q, \vec{\mathbf{v}}_d)$ is equal to the projection of $-B\vec{\mathbf{v}}_d$ onto the nearest side of the friction cone at q (the projection is taken normal to the axis of the cone, see Figure 3.3(c));
- if $q \in \mathcal{C}_{contact}$, q is at a contact vertex, then the reaction force is a positive linear combination of the reaction forces that can be exerted by the edges abutting the vertex.

According to the above definition, if $\vec{\mathbf{v}}_d^c$ and θ are such that all directions in the control uncertainty cone project positively on the outgoing normal of a surface, the robot is guaranteed to move away from the surface. If the control uncertainty cone is completely included in the inverted friction cone, then the robot is guaranteed to stick on the surface. If all directions in the control uncertainty cone project positively on the ingoing normal of the surface but none of them lies in the inverted friction cone, then the robot is guaranteed to slide along the surface. In any other case the behavior of the robot cannot be predicted at planning time.

3.3.3 Sensing

Real robots are equipped with a variety of sensing equipment that provide measurements of the actual configuration of the robot and the workspace. These measurements contain stochastic errors, whose distribution is called *sensing uncertainty*. Clearly, the more information the robot has, the more complex tasks it is able to perform. On the other hand, though, reasoning about sensing information at planning time is a significant source of complexity for planners since the actual sensed values are not known at planning time. It is therefore important to distinguish the information that is more likely to have a significant impact on the structure of a plan, and ignore other potential sources of less important information. As a result, researchers have been particularly frugal when selecting the sensing inputs that a planner-with-uncertainty considers.

Sticking This is the simplest form of sensing, and for this reason robots that use it are often called *sensorless*. The only thing that the robot can sense is its own motion as a binary variable (1 if it is in motion, 0 if it is stopped). The robot moves following the generalized damper model, using a termination condition that stops its motion automatically (sticking on obstacle edges due to friction). Nevertheless, the robot must realize that motion has stopped, in order to proceed with the execution of the next command of the plan. It is assumed that the robot never makes a mistake in deciding whether it moves or not, so no sensing uncertainty exists in this case.

Force sensing A force sensor measures the reaction force exerted on the robot. The output of the sensor is a vector denoted by $\vec{\hat{\mathbf{f}}}$. For simplification, we assume that the modulus of the measured force is accurate. On the other hand, the magnitude of the angle between $\vec{\mathbf{f}}$ and $\vec{\hat{\mathbf{f}}}$ (if they are non-zero vectors) is less than a fixed angle $\varepsilon < \pi/2$. Hence, $\vec{\mathbf{f}}$ lies in an open cone of angle 2ε whose axis points along $\vec{\hat{\mathbf{f}}}$.

Position sensing A position sensor measures the current configuration of the robot. The sensed configuration is denoted by $\hat{q}$, while the current actual configuration is denoted by q. The uncertainty in the measurement is modeled as an open disc $\mathcal{U}_q(\hat{q}) \subset \Re^2$ of fixed radius ρ centered at the sensed configuration $\hat{q}$. This means that if the position sensor returns $\hat{q}$, the current actual configuration q may be anywhere in $\mathcal{U}_q(\hat{q}) \cap \mathcal{C}_{feasible}$. Reciprocally, if the actual configuration is known to be q, the sensed configuration $\hat{q}$ should necessary lie in $\mathcal{U}_q(q)$.[2]

3.4 Preimage backchaining in two dimensions

3.4.1 Statement of the problem

Let $\mathcal{I}$ be a subset of $\mathcal{C}_{feasible}$, in which it is known at planning time that the robot's configuration will be when the execution of the motion plan starts. $\mathcal{I}$ is called the *initial region*. Let $\mathcal{G}$ be another subset of $\mathcal{C}_{feasible}$ called the *goal region*. Given a robot that can be modelled as described in the previous sections, we want the planner to generate a sequence of motion commands (a linear motion plan), whose execution makes the robot move from its actual initial configuration in $\mathcal{I}$ to a final configuration in $\mathcal{G}$.

3.4.2 Preimage backchaining

The plan is generated by working backwards from the goal (backchaining). During each step, we identify a target set $\mathcal{T}$ (initially the goal), we select a motion command $(\vec{\mathbf{v}}_d^c, \mathbf{tc})$, and we compute the set of configurations $\mathcal{P}(d, \mathbf{tc}; \mathcal{T})$ from where the motion command is guaranteed to make the robot reach the target set (reachability) and stop in it (recognizability). $\mathcal{P}(d, \mathbf{tc}; \mathcal{T})$ is the preimage of $\mathcal{T}$ as defined in Chapter 2. If the initial region is covered entirely by the preimage, then the motion command $(\vec{\mathbf{v}}_d^c, \mathbf{tc})$ is guaranteed to bring the robot into $\mathcal{T}$ and make it stop in there. If this is

---

[2] This is not always true. It happens to be so in this case because the perception function $\mathcal{U}_q(\hat{q})$ is symmetric with respect to $\hat{q}$ (i.e., $x \in \mathcal{U}_q(\hat{q}) \equiv 2\hat{q} - x \in \mathcal{U}_q(\hat{q})$). If this weren't so, the interpretation function $\mathcal{U}_q^{-1}(q)$ would be the symmetric set of $\mathcal{U}_q(q)$ with respect to q.

the case, the algorithm terminates; otherwise, $\mathcal{P}$ becomes the target set of the next backchaining step. The number of motion commands in a plan is equal to the number of backchaining steps.

The computation of preimages combines the notions of reachability and recognizability. Reachability relates to the fact that any trajectory obtained by executing a motion commanded along $\vec{\mathbf{v}}_d^c$ from a preimage of $\mathcal{T}$ should reach $\mathcal{T}$. Given a starting configuration, $\vec{\mathbf{v}}_d^c$ only determines a *commanded trajectory* τ_d^c. Due to errors in control, any execution produces an *actual trajectory* τ_d that is slightly different. The planner must be certain that all the possible actual trajectories consistent with both $\vec{\mathbf{v}}_d^c$ and control uncertainty (i.e., θ) will reach $\mathcal{T}$ at some instant.

Reaching $\mathcal{T}$, however, is not sufficient since the termination condition could just let the robot traverse the target set without stopping in it, or, instead, it could halt the motion prematurely before $\mathcal{T}$ has been reached. In fact, the planner must be certain that the termination condition `tc` will stop the robot in $\mathcal{T}$ (recognizability). The function `tc` is an observer of the actual trajectory τ_d being executed. Since sensing is imperfect, `tc` perceives τ_d as an *observed trajectory* $\hat{\tau}_d$ which is most likely to be neither the commanded one, nor the actual one. Thus, the problem for the planner is to be sure that, for every possible observed trajectory $\hat{\tau}_d$, (a) `tc` becomes `true` at some instant, and (b) at the instant t_f when `tc` first becomes `true`, all the actual trajectories τ_d consistent with $\hat{\tau}_d$ (i.e., the trajectories which may be observed as $\hat{\tau}_d$ given the uncertainty in sensing) have reached the target set, namely, $\tau_d(t_f) \in \mathcal{T}$.

Preimage computation has been investigated in depth in [32, 52]. For a given commanded velocity $\vec{\mathbf{v}}_d^c$, the ideal method would compute the maximal preimage of $\mathcal{T}$ over all possible termination conditions, i.e., a preimage that is not contained in any other preimage[3] of $\mathcal{T}$ for $\vec{\mathbf{v}}_d^c$. The method would also return the termination condition for the maximal preimage. Intuitively, a large preimage has more chance to include the initial region $\mathcal{I}$ than a small one; in addition, if it is considered recursively as an intermediate goal, a large preimage has more chance to admit larger preimages than a small one. Thus, considering larger preimages may reduce the size of the search

---

[3] By definition, any subset of a preimage is also a preimage of the same set for the same motion command.

graph; it may also produce "simpler" strategies (strategies made up of less motion commands). Nevertheless, Erdmann [32] showed that: (a) a maximal preimage does not always exist; (b) if one does exist, it may not be unique; and (c) if a unique maximal preimage exists, it may depend in a very subtle fashion on sensing history, the elapsed time since the beginning of the motion, and the knowledge embedded in the termination condition.

Most of the difficulties related to maximal preimages are due to the subtle interdependence between reachability and recognizability. Indeed, even after a particular termination condition has been selected, the region of the workspace where this condition always evaluates to true (the kernel of the termination condition as defined in Chapter 2), which is needed in order to compute the preimage, may depend on the region from where this command is executed, which is precisely the preimage of this region.

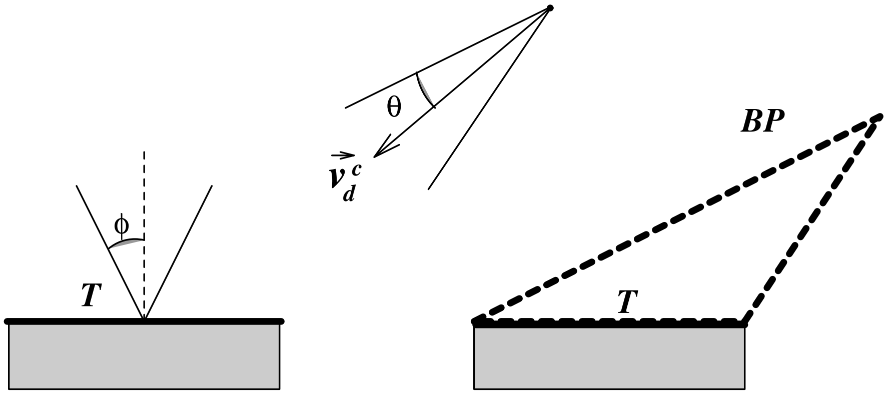
3.4.3 Backprojection from target kernel

One way of breaking this recursive dependence is to use restricted forms of the termination condition so that these two issues can be treated separately. The basic idea is to identify a subset $\mathcal{K}$ of the target set $\mathcal{T}$, in which achievement of $\mathcal{T}$ can be recognized independently of the starting region. A termination condition is constructed, so that it always evaluates to `true` when the robot is in $\mathcal{K}$, i.e., $\mathcal{K}$ is the kernel of this termination condition according to definition 2.3. In this case, though, we first compute $\mathcal{K} = \chi(d; \mathcal{T})$ as a function of the target set and possibly the commanded direction of motion, and we construct the termination condition that corresponds to $\mathcal{K}$ afterwards. It now remains to compute the region $\mathcal{BP}(d; \mathcal{K})$, from where the robot is guaranteed to attain the kernel if the commanded velocity is $\vec{\mathbf{v}}_d^c$. In Chapter 2 we call this region the backprojection of $\mathcal{K}$.

For every possible termination condition, the backprojection of a set is bigger than (or equal to) any preimage of the set for the same commanded direction of motion:

$$\forall d \, \forall \mathbf{tc} \;\; \mathcal{P}(d, \mathbf{tc}; \mathcal{T}) \subseteq \mathcal{BP}(d; \mathcal{T}).$$

For example, look at Figure 3.4. Assume that the robot can only sense sticking on



(a) (b)

Figure 3.4: Preimage and backprojection

obstacle edges. Its target set $\mathcal{T}$ is the top edge of a rectangular box. For choices of $\vec{\mathbf{v}}_d^c$ such that the control uncertainty cone is not contained in the inverted friction cone, the preimage of $\mathcal{T}$ is the empty set since the robot cannot stick on the target edge. On the other hand, the backprojection of $\mathcal{T}$ is outlined with the thick dashed line in Figure 3.4(b).

Conversely, there exists a termination condition **tc**, for which the preimage of a set is bigger than (or equal to) the backprojection of the kernel of the set for the same commanded direction of motion:

$$\forall d\, \exists \mathbf{tc}\ \ \mathcal{P}(d, \mathbf{tc}; \mathcal{T}) \supseteq \mathcal{BP}(d; \chi(d; \mathcal{T})).$$

This follows directly from the definition of the kernel. Whenever the termination condition $\hat{q} \in \mathcal{U}_q(\chi(d, \mathcal{T}))$ is true, we know that $q \in \mathcal{T}$. The preimage of $\mathcal{T}$ for this termination condition is equal to the backprojection of the kernel. Other termination conditions may produce bigger preimages.

3.5 Kernel computation

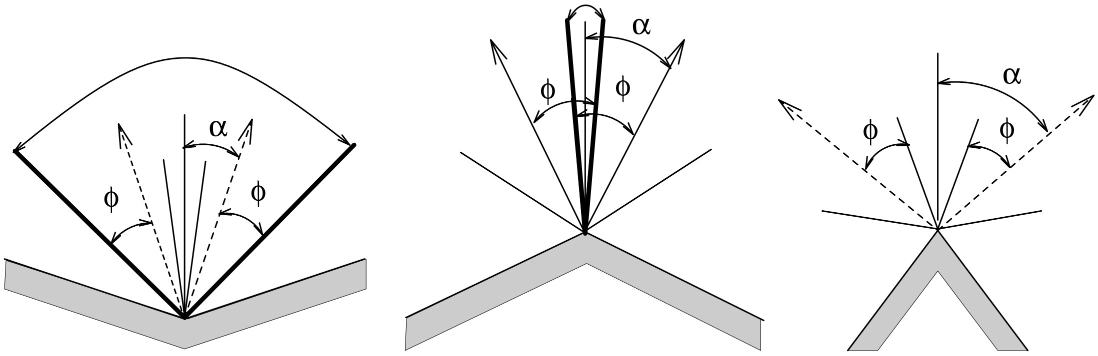
The computation of the kernel of a target set depends on the particular sensing readings that are being provided to the termination condition. In this section we describe kernel computation for all the sensing schemes presented in Section 3.3 (sticking, force sensing, position sensing), as well as for the combination of all three together.

3.5.1 Sticking

When sticking on obstacles is the only thing that can be sensed, there is no sensing uncertainty: once the robot sticks, motion terminates. Whether the robot will stick or not on an obstacle edge depends on its actual direction of motion at execution time. Since we are interested in producing guaranteed plans, the planner must identify those edges and vertices where the robot is guaranteed to stick (the *sticking edges/vertices*). Therefore, the planner must make sure that all actual velocity vectors that are compatible with the commanded velocity $\vec{\mathbf{v}}_d^c$ make the robot stick on such an edge or vertex.

The robot is guaranteed to stick on an edge, if the inverted control uncertainty cone lies entirely inside the friction cone of the edge. However, when the robot hits a vertex the situation is more complicated. We previously assumed that a vertex can exert a reaction force that is a positive linear combination of the forces exerted by the two edges forming the vertex. Consequently, the friction cone of a vertex consists of the friction cones of the two edges and the region between them. If the inverted velocity vector of the robot is included in the vertex friction cone, then the robot sticks at the vertex. However, small perturbations in the position of the robot may cause it to move on one of the two edges that form the vertex, and then move according to the reaction force that is exerted by this edge only. If this force pushes the robot back towards the vertex on both edges or makes it stick on the edge, then sticking at the vertex is stable; otherwise, the robot will move away from the vertex. We are only interested in stable sticking. The *sticking cone* of a vertex consists of the inverted velocity directions of the robot that cause stable sticking at the vertex.

Consider the friction cones (anchored at the vertex) of both edges. The region between each edge and its friction cone contains all inverted velocity directions that push the robot towards the vertex. The union of this region with the friction cone creates a bigger cone that contains all inverted commanded velocity directions that cause the robot either to stick at the vertex or to slide on the edge towards the vertex. The sticking cone of the vertex is the intersection of these cones of the two edges. In Figure 3.5 we show the sticking cones of three vertices (the region outlined with thick black rays): (a) For a concave vertex, the sticking cone is always equal to the friction



(a) (b) (c)

Figure 3.5: The sticking cone of a vertex

cone; (b) for a convex vertex, the sticking cone is the intersection of the friction cones of the two edges; but (c) when this intersection is empty, the sticking cone is null.

Given a commanded velocity $\vec{\mathbf{v}}_d^c$ and a target set $\mathcal{T}$, the computation of the kernel is done by (a) identifying all edges and vertices in contact space that belong to $\mathcal{T}$, (b) selecting those edges whose friction cone includes the inverted control uncertainty cone for $\vec{\mathbf{v}}_d^c$, and (c) selecting those vertices whose sticking cone includes the inverted control uncertainty cone for $\vec{\mathbf{v}}_d^c$. More specifically, the kernel consists of every edge E in $\mathcal{T} \cap \mathcal{C}_{contact}$ such that $|angle(-\vec{\mathbf{v}}_d^c, \vec{\nu}(E))| \leq \phi - \theta$, and every vertex V in $\mathcal{T} \cap \mathcal{C}_{contact}$ such that, if V is concave, $|angle(-\vec{\mathbf{v}}_d^c, \vec{\nu}(V))| \leq \phi + \alpha - \theta$, and if V is convex, $|angle(-\vec{\mathbf{v}}_d^c, \vec{\nu}(V))| \leq \phi - \alpha - \theta$. Here $\vec{\nu}(V)$ is the bisector of the angle of the normals of the two edges, and α is half the angle between these normals.

Notice that guaranteed sticking on edges is possible only if $\phi > \theta$, i.e., the friction cone is larger than the control uncertainty cone. This is not required for sticking at concave vertices. In fact, sticking at a concave vertex is easier and more stable than sticking along a contact edge, since the vertex may be reached by sliding along the two abutting edges.

3.5.2 Force sensing

The force sensor measures the reaction force exerted by an obstacle edge on the robot. Let $\mathbf{F}_{reac}(q, d)$ denote the set of all possible reaction forces at a configuration

q, when the commanded velocity is $\vec{\mathbf{v}}^c_d$. If, for example, the inverted velocity cone lies completely outside the friction cone, and it projects positively on the outgoing normal on the surface, then $\mathbf{F}_{reac}(q, d)$ consists of a single vector along one extreme ray of the friction cone. Let $\hat{\mathbf{F}}_{reac}(q, d)$ denote the set of all possible sensed forces at a configuration q, when the commanded velocity is $\vec{\mathbf{v}}^c_d$. $\hat{\mathbf{F}}_{reac}(q, d)$ is derived from $\mathbf{F}_{reac}(q, d)$ as follows:

- If $0 \in \mathbf{F}_{reac}(q, d)$, then $0 \in \hat{\mathbf{F}}_{reac}(q, d)$;
- If $\mathbf{F}_{reac}(q, d)$ contains a non-zero vector $\vec{\mathbf{f}}$, then all vectors $\vec{\hat{\mathbf{f}}}$ such that $\|\vec{\hat{\mathbf{f}}}\| = \|\vec{\mathbf{f}}\|$ and $|angle(\vec{\mathbf{f}}, \vec{\hat{\mathbf{f}}})| < \varepsilon$ are in $\hat{\mathbf{F}}_{reac}(q, d)$.

Computing the set $\hat{\mathbf{F}}$ for each edge and its intersection with other such sets is straightforward and takes constant time.

Force sensing is not able to distinguish points in the same edge, it can only recognize that the robot is in contact with some edge whose orientation is compatible with the sensed reaction force. If the reaction force on an edge may be null, this edge cannot be distinguished from the free space. Thus, if only force sensing is available, the kernel of a target set for a given commanded velocity consists of all contact edges that are entirely in the target and can neither generate a force measurement that could have been generated by an edge that does not belong to the target set, nor a null measurement.

3.5.3 Position sensing

In the case of position sensing, the sensory readings at a particular configuration do not depend either on the commanded or on the actual velocity. If the execution module makes use of only position sensing readings to decide whether the robot is in the goal or not, the computation of the kernel is conceptually simple: The kernel consists of all points of the target set $\mathcal{T}$ that cannot be confused with any point that does not belong to $\mathcal{T}$.

Two points of the configuration space are confusable if they may produce the same sensory reading. According to our position sensing uncertainty model, the distance

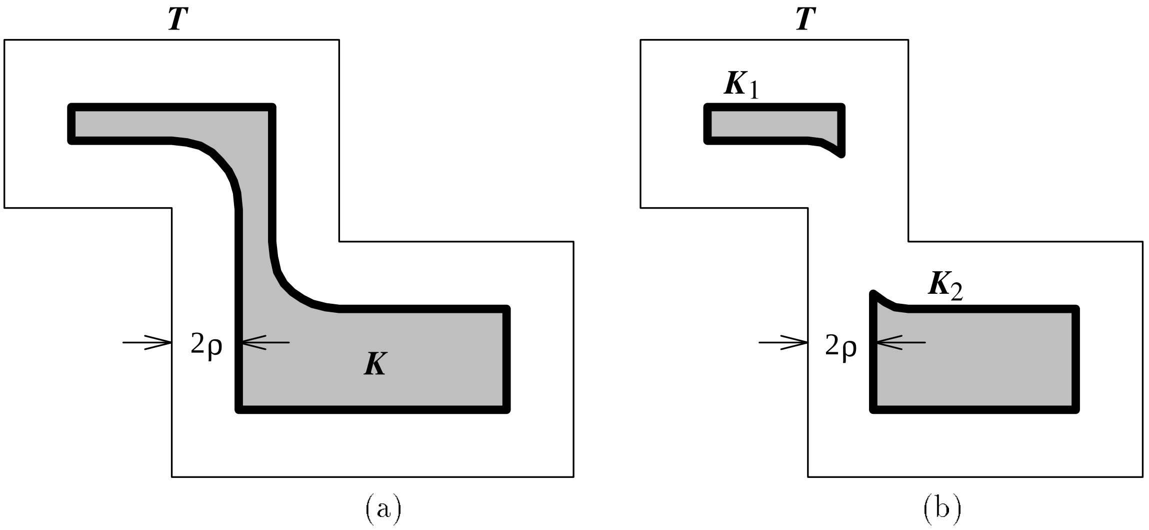


Figure 3.6: Computing the kernel in free space

between a point in the configuration space and the sensed configuration, when the robot lies at this point, is less than ρ, the position sensing uncertainty. Therefore, only points that are closer than 2ρ may produce the same sensory reading (and thus be confusable). In order to identify the points that are confusable with feasible configurations but do not belong to $\mathcal{T}$, we grow $\mathcal{C}_{feasible} \setminus \mathcal{T}$ by 2ρ. All the points of $\mathcal{T}$ that are not in this set constitute the kernel of $\mathcal{T}$.

In general, a target set $\mathcal{T}$ consists of polygonal regions, edges and points in $\mathcal{C}_{feasible}$. Edges and points are treated as degenerate polygons, so no special algorithms are needed to handle them. If a target region $\mathcal{T}$ lies in the free space without intersecting any other target regions, then its kernel $\mathcal{K}$ is found by shifting its edges inwards by 2ρ and drawing a circle of radius 2ρ at concave vertices. These lines are connected to form the kernel of the region (Figure 3.6). Of course, if $\mathcal{T}$ is thinner than 4ρ its kernel is the empty set. Also, the kernel of a connected set may be not connected, as is evident in the example of Figure 3.6(b).

If two or more target regions in the free space touch or overlap, we need to compute their union as a single set before we call the above algorithm. As an example, consider two adjacent rectangular regions. Their common edge should not be shifted inwards, as long as the robot does not need to know which of the two regions achieves. Indeed,

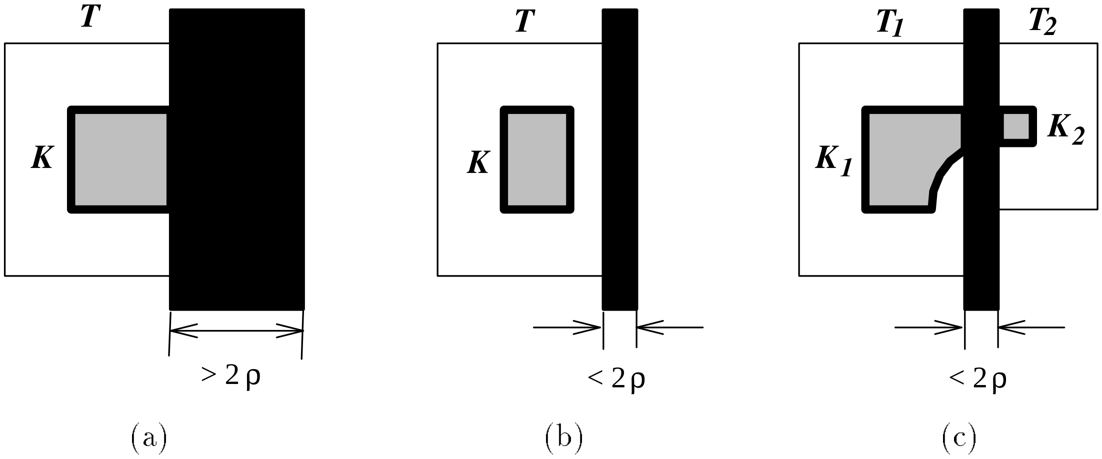


Figure 3.7: Computing the kernel in contact with obstacles

the points of the two regions whose distance from the common edge is smaller that 2ρ cannot be distinguished from points in the other region, but they are not confusable with points which lie outside both regions.

Another interesting case occurs when a target region is in contact with some obstacle. The interior points of the obstacle are not feasible, therefore we do not mind if they are confusable with points in the kernel. The general idea is that the edges of a target region that are in contact with obstacles should not be shifted inwards (Figure 3.7(a)). However, if the obstacle is too thin (its thickness is smaller than 2ρ), there are free space points that are confusable with points of the target set near the contact edge. In such a case, we compute the kernel by shifting inwards all target and obstacle edges that are not in contact with each other (Figure 3.7(b)). The same technique can be used to handle the case where another target region lies on the other side of the obstacle, and it is in contact with the obstacle (Figure 3.7(c)).

To formalize these ideas, we introduce the notion of the *cylsphere*. The cylsphere of an edge is the set of points whose distance from the edge is less than a real number r: $cylsphere(E, r) = \{q \in \mathcal{C} \mid dist(q, E) < r\}$. In order to compute the kernel $\mathcal{K}$ of a collection of target polygonal regions $\mathcal{T}_i$, we perform the following steps:

1. $\mathcal{K} = \emptyset$.

2. Select a target region $\mathcal{T}_i$.

3. Construct the maximal connected polygonal region S as follows: $\mathcal{T}_i$ is first included in S; then, every connected component of $\mathcal{CB}$ and target polygon $\mathcal{T}_j$, $j \neq i$, that shares an edge with a region already in S is iteratively[4] included in S, and it is marked as visited.

4. $S' \leftarrow S$.

5. For every edge E in the boundary of S, $S' \leftarrow S' \setminus cylsphere(E, 2\rho)$.

6. For all target regions $\mathcal{T}_j \in S$, add $\mathcal{T}_j \cap S'$ to the kernel $\mathcal{K}$.

7. If all target regions have been visited, return $\mathcal{K}$; otherwise, go back to step **2**.

Let $O(n)$ be the combined complexity of $\mathcal{C}_{contact}$ and $\mathcal{T}$. The region constructed at step **3** has $O(n)$ edges. Using a sweep-line algorithm, step **3** can be performed in $O((n+c)\log n)$ time, where $c \in O(n^2)$ is the number of intersections among target region edges. Step **5** can be performed in the same bound, only in this case c is the number of intersections among cylspheres. $\mathcal{T}_j \cap S'$ can be obtained from the sweep algorithm with the same time complexity.

Once the kernel has been computed, the remaining task is to extract a termination condition that is guaranteed to be true when the robot is in the kernel and false when the robot is outside the target set (so that the robot does not stop before reaching the target set). If $\hat{q}$ is the sensed configuration of the robot, the termination condition that corresponds to kernel $\mathcal{K}$ is $\mathbf{tc}(\hat{q}) \equiv dist(\hat{q}, \mathcal{K}) < \rho$. Equivalently, the robot terminates its motion whenever $\hat{q}$ lies within a set that we get by isotropically pushing out the boundary of the kernel by ρ. By construction of the kernel, all actual configurations that are compatible with sensory readings in this set lie in the target set. Figure 3.8 shows an example with a single target region $\mathcal{T}$ in contact with a C-obstacle $\mathcal{CB}$. The kernel $\mathcal{K}$ is the inner darker region; the set that contains the sensory readings that make the termination condition evaluate to true is outlined with the thin dashed line; and the set of configurations where the robot may stop is outlined with the thicker dashed line. Note that this set is a proper subset of the target set.

---

[4] In other words, S is the transitive closure of all connected components of $\mathcal{CB}$ and target polygons touching $\mathcal{T}_i$.

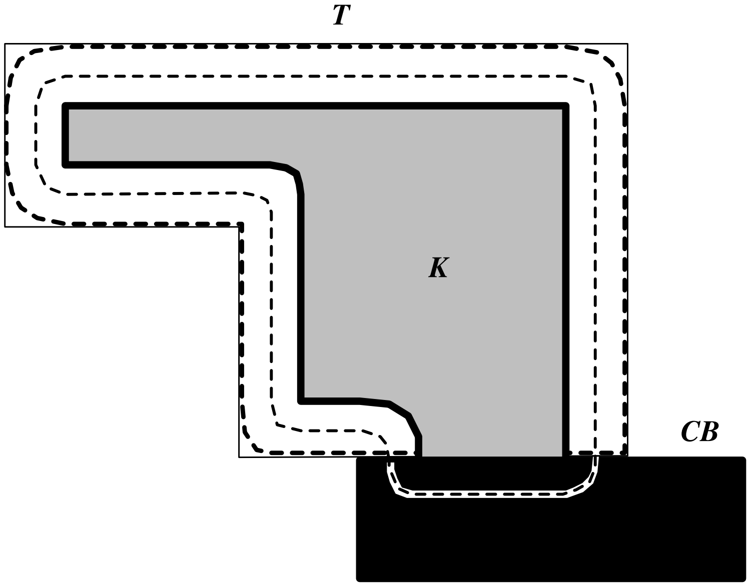


Figure 3.8: The kernel and the termination condition

3.5.4 Combination of sensing

Usually, robots have both position and force sensors, and can detect sticking, too. We now investigate how to compute the kernel for the combination of all three sensing schemes.

Sticking does not entail recognizing configurations, thus it does not interact with the two other kinds of sensing; it merely contributes a number of sticking edges to the kernel.

Position and force sensing do interact and create kernels that are bigger than the union of the kernels computed individually for each sensor. Such points, that are in neither individual kernel, but are in the combined kernel, must be in contact space so that both sensors can be used. Indeed, points in contact space can now be recognized using both position and force sensing, if they are either more than 2ρ apart, or if they cannot generate the same reaction force. This computation can be done as follows:

For each contact edge $E \in \mathcal{T}$:

1. If $\hat{\mathbf{F}}_{reac}(E, d)$ contains the null vector, then the edge is discarded.

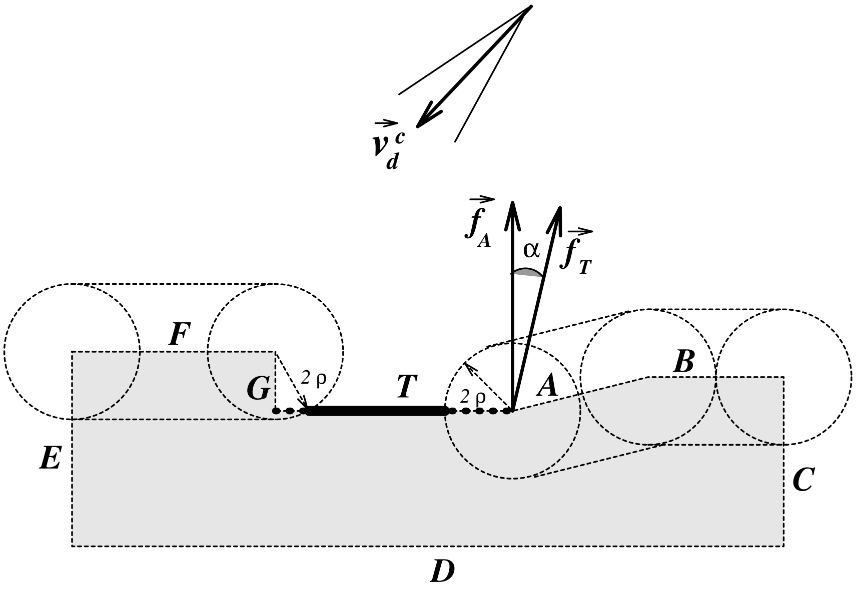


Figure 3.9: The kernel of a target in contact space

2. Otherwise, $S \leftarrow E$.

3. For every contact edge $C \notin \mathcal{T}$:
If $\hat{\mathbf{F}}_{reac}(E, d) \cap \hat{\mathbf{F}}_{reac}(C, d) \neq \emptyset$, then $S \leftarrow S \setminus cylsphere(E, 2\rho)$.

4. S is added to the kernel.

Consider the example of Figure 3.9. The goal is a single edge T of a C-obstacle that is depicted in light grey for clarity. The commanded velocity is such that the inverted control uncertainty cone is outside the friction cones of edges F, T, A and B, and projects positively on the outgoing normal of these edges (i.e., the robot is guaranteed to slide on these edges). The robot is also guaranteed to stick on C and G, and move away from D and E.

The sticking termination condition yields no kernel edges, because no sticking is possible on T. Using the above algorithm, we visit all edges in contact space, and if the reaction forces they may produce are confusable with the reaction force that may be produced by T, we subtract their cylspheres from T. The only reaction force $\vec{\mathbf{f}}_T$ that can be generated by T lies along the right ray of the friction cone. This cannot be confused with the forces generated by D and E (zero force), or C and G ($\vec{\mathbf{f}}_T$ lies

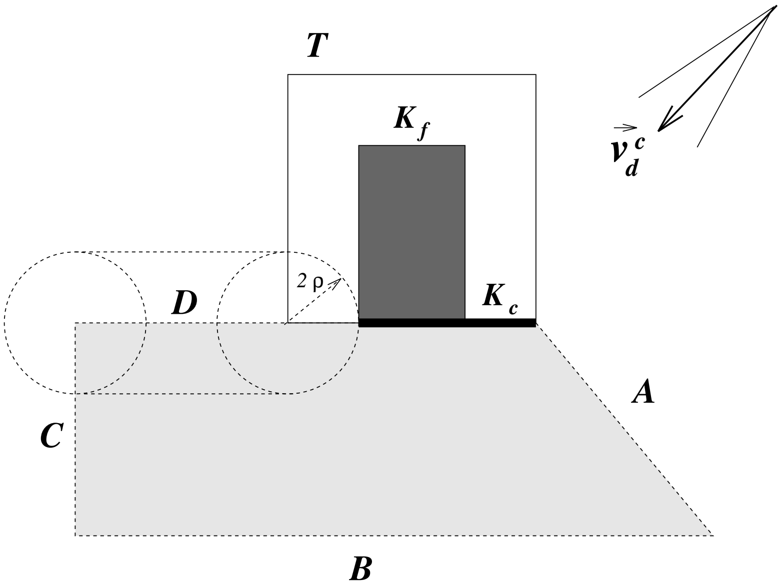


Figure 3.10: The kernel of a target lying both in free and in contact space

outside the friction cone of C or G). Edges F and B are parallel to T so they generate the same force $\vec{\mathbf{f}}_T$. They are confusable with T, and their cylspheres of radius 2ρ are subtracted from T. While B is too far away to have any effect, the cylsphere of F cuts a small part of T's left end. Edge A can also generate a single reaction force $\vec{\mathbf{f}}_A$. Since the angle α between $\vec{\mathbf{f}}_T$ and $\vec{\mathbf{f}}_A$ is smaller than the force uncertainty ε, edges T and A are confusable relative to force sensing. The cylsphere of A cuts a piece of length 2ρ from the right part of T. The remaining part of T, depicted as a solid black line in the figure, is the desired kernel.

As a further example consider Figure 3.10. Here, the target region $\mathcal{T}$ lies both in free and contact space. Its kernel consists of the free space set $\mathcal{K}_f$ and of the contact edge $\mathcal{K}_c$. Note that neither $\mathcal{K}_c$ is included in $\mathcal{K}_f$, nor $\mathcal{K}_f$ is included in $\mathcal{K}_c$. This means that by considering both position and force sensing simultaneously we create larger kernels (and consequently preimages) than either of them would produce on its own.

3.6 Computation of backprojections

After the kernel has been computed as a collection of free space regions and contact edges and vertices, we wish to compute its backprojection in order to find out whether it includes the initial region or not.

An algorithm for this task computes the backprojection of each kernel component independently and then merges them in a second pass [32, 52]. This algorithm has been generalized by Canny and Donald [23] to a one-pass algorithm that generates the backprojection of any region $\mathcal{K}$ described as the union of contact edges and polygonal regions in $\mathcal{C}_{feasible}$. The algorithm sweeps a line L across the plane perpendicularly to the commanded velocity $\vec{\mathbf{v}}_d^c$. The sweep starts at a position of L where it is tangent to $\mathcal{K}$, with $\mathcal{K}$ lying entirely on the side of L pointed to by the vector $-\vec{\mathbf{v}}_d^c$. The sweep proceeds in the direction of the vector $-\vec{\mathbf{v}}_d^c$. During the sweep, the algorithm maintains the status of L, i.e., the description of its intersection with $\mathcal{C}_{contact}$, $\mathcal{K}$, and rays parallel to the edges of the inverted velocity cone that are erected from contact vertices and goal vertices. This status changes at events occurring when L passes through a vertex of $\mathcal{C}_{contact}$, a vertex of $\mathcal{K}$, the intersection of two rays, the intersection of a ray and $\mathcal{C}_{contact}$, or the intersection of a ray and $\mathcal{K}$. At each of these events, the algorithm updates the status of the line and the list of future events. During the sweep, the algorithm traces out the backprojection. The sweep stops when the last event is encountered.

The key idea of this algorithm is that the backprojection's boundary consists of straight segments, and that at every event there exists a local criterion (which does not depend on what will happen later during the sweep) for deciding how to trace the backprojection's boundary. This criterion leads to erecting rays from both contact and goal vertices, in order to prevent the backprojection from containing configurations from where a motion may reach a vertex or an edge, where it may stick or slide away from the goal with no possibility of returning to it. The algorithm does not require the backprojection to be bounded, but it does require that the actual velocity of the robot never project negatively into the $\vec{\mathbf{v}}_d^c$ direction. This condition is achieved if the friction cone is larger than the control uncertainty cone, i.e., $\phi \geq \theta$.

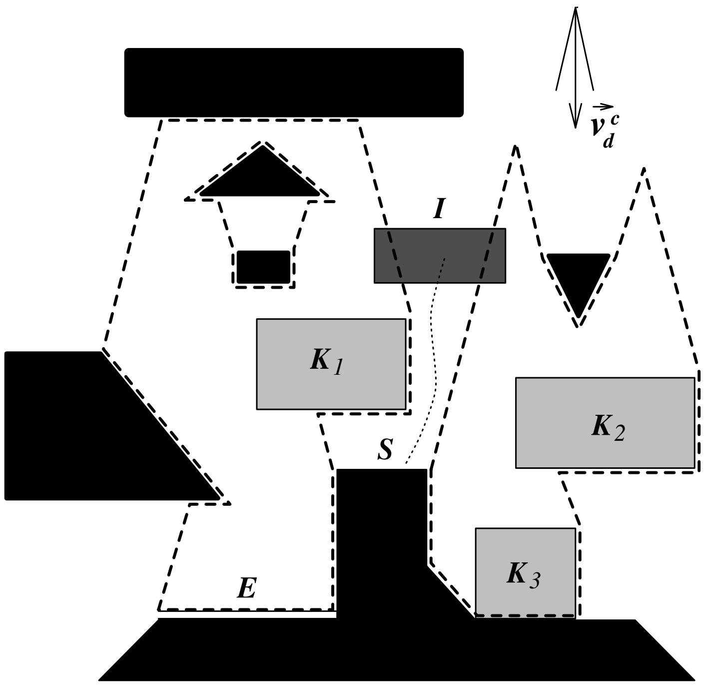


Figure 3.11: Computation of Backprojection

Let n be the number of vertices in $\mathcal{C}_{contact}$. Assume that the combined complexity of $\mathcal{C}_{contact}$ and $\mathcal{K}$ (the number of vertices in $\mathcal{C}_{contact}$ plus the number of vertices of $\mathcal{K}$ plus the number of intersections between $\mathcal{K}$ and $\mathcal{C}_{contact}$) is $O(n)$. The sweep-line algorithm erects $O(n)$ rays. The contour of the backprojection consists of goal edges (edges bounding $\mathcal{K}$), rays in free space and contact edges. Each erected ray can contribute at most one edge to the backprojection. Each goal edge contributes at most one backprojection edge. Each non-goal edge in contact space can contribute one or several backprojection edges, but whenever it contributes k backprojection edges, it also "consumes" $O(k)$ edges in free space. Hence, the boundary of the backprojection consists of $O(n)$ edges. Since each ray is interrupted at its first intersection with a goal edge, a contact edge, or another ray, the backprojection can be computed in $O(n \log n)$ time.

Figure 3.11 illustrates the backprojection of a kernel that consists of three regions K_1, K_2, K_3 and an edge E. The black polygons are C-obstacles. The backprojection (outlined with a dashed line) consists of two disconnected components, one of which contains a hole. Since the initial position (the grey rectangular region $\mathcal{I}$) is not completely contained in the backprojection, the robot is not guaranteed to reach the kernel if it starts from $\mathcal{I}$ and follows $\vec{\mathbf{v}}_d^c$. Indeed, it is possible for the robot to stick on vertex S and never reach the kernel.

3.7 Implementation and experimentation

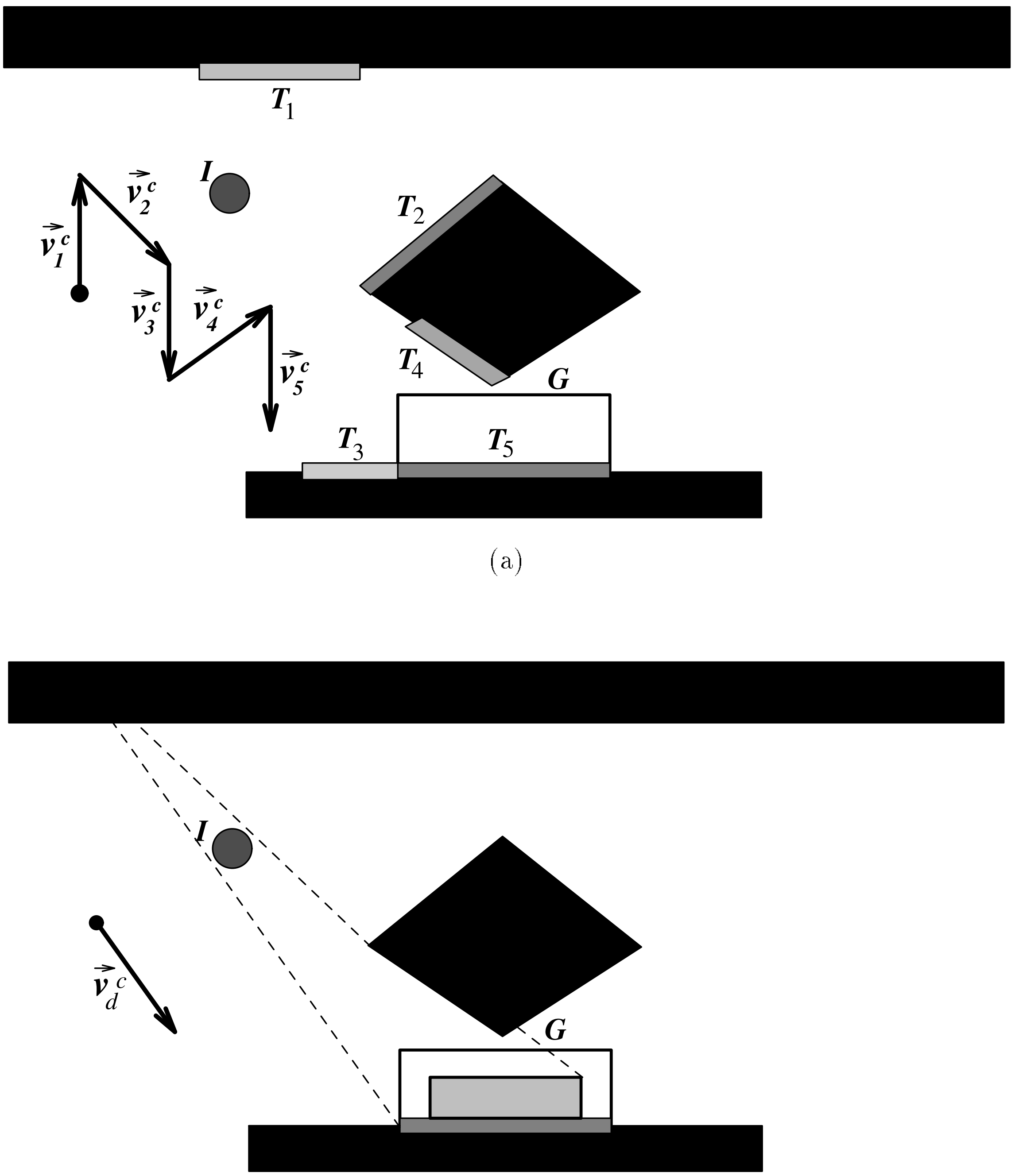
We have implemented a motion planner based on the preimage backchaining approach (for details see [52, 54, 55]). This planner computes preimages using either the sticking termination condition, or the combination of position and force sensing, or all three together. The user inputs the description of the configuration space, the goal region $\mathcal{G}$, and the initial region $\mathcal{I}$. If successful, the planner returns a motion plan in the form of a sequence of unit commanded velocities and the associated sequence of computed preimages. The method used for computing the preimages determines the termination condition of every motion command in the plan. The planner is implemented in Allegro Common Lisp and runs on an Apple Macintosh II computer.

The planner constructs a graph of preimages by considering commanded velocities with the orientations $\{k\pi/N\}_{k=0,\ldots,2N-1}$, where N is input by the user. In our experiments, we used $N = 2$ or 4. The program searches the graph in a breadth-first fashion, but other (and perhaps more efficient) search techniques could have been used as well.

The algorithms implemented in the planner are essentially those described in the previous sections, with minor variations. In particular, the backprojection of a region is computed using an algorithm similar to that described in [52], rather than the sweep-line algorithm (which might be faster). The planner also approximates conservatively the generalized polygonal kernels by polygonal regions. These changes have no major impact on the visible operations of the planner.

In all the examples shown, the initial region is a single point; in the figures, it is the center of a disc that depicts position uncertainty. The control uncertainty cone and the force uncertainty cone are not depicted.

Figure 3.12(a) illustrates a plan generated with the sticking termination condition.



(a)

(b)

Figure 3.12: Example 1

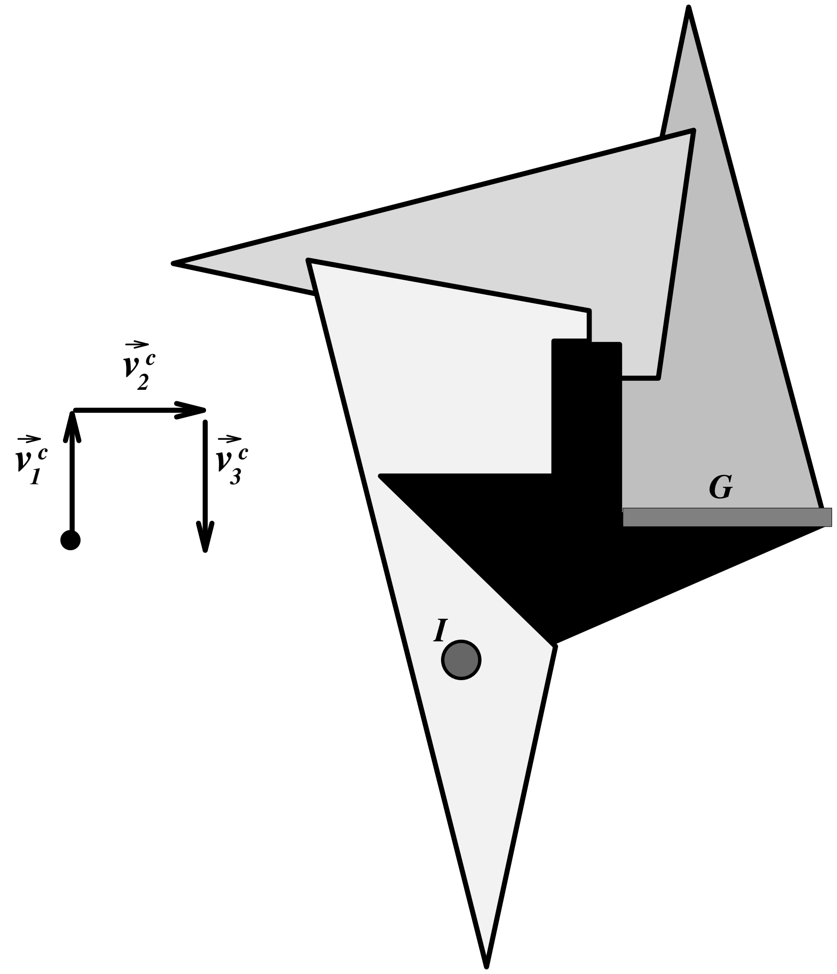


Figure 3.13: Example 2

The computed preimages are not shown in the figure. The goal is the rectangular region $\mathcal{G}$. The plan constructed consists of 5 steps, defined by the commanded velocities $\vec{\mathbf{v}}_1^c$ through $\vec{\mathbf{v}}_5^c$. The corresponding sticking edges are $\mathcal{T}_1$ through $\mathcal{T}_5$. ($\mathcal{T}_i$ is the intersection of the preimage of $\mathcal{T}_{i+1}$ and the sticking subset of $\mathcal{C}_{contact}$ for $\vec{\mathbf{v}}_i^c$. A motion commanded along $\vec{\mathbf{v}}_i^c$ and issued from within $\mathcal{T}_{i-1}$ is guaranteed to terminate in $\mathcal{T}_i$, but some configurations in $\mathcal{T}_i$ may not be reachable.) This plan is typical of the "bouncing on obstacles" that is caused by the sticking termination condition. On the other hand, if we use position and force sensing, the resulting plan consists of a single motion command along $\vec{\mathbf{v}}_d^c$ (Figure 3.12(b)).

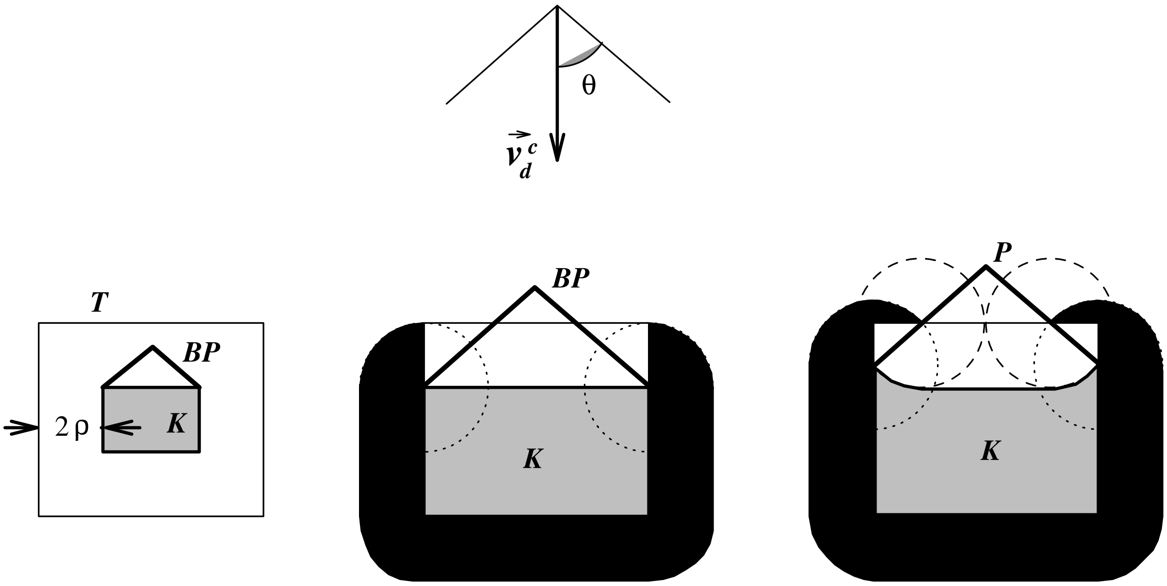
Figure 3.13 shows an example which cannot be solved by using sticking on obstacle edges. The goal is just one obstacle edge marked with G in the figure. In this case, there are no edges facing each other so that the robot can bounce between them. The combination of position and force sensing creates the three-step plan shown in the figure.

3.8 On the termination condition's knowledge

We now return to the computation of the kernel in the case where only position sensing is available, and examine how the method described in this chapter relates to the generic method described in Chapter 2. Consider the simple case where the target set $\mathcal{T}$ is a rectangular region of the free space. According to the method described in this chapter its kernel $\mathcal{K}$ is found by shrinking $\mathcal{T}$ by 2ρ. Then $\mathcal{T}$'s preimage is computed as the backprojection of $\mathcal{K}$ (Figure 3.14(a)). The termination condition (i.e., the set of all sensory values for which the robot will stop) is found by isotropically growing the kernel by ρ. (In Chapter 2 we called this set $\hat{S}_{\mathbf{tc}}$.) The set of all configurations that are compatible with the termination condition ($\mathbf{INT}(\hat{S}_{\mathbf{tc}})$, according to the terminology of Chapter 2) is found by isotropically growing the kernel by 2ρ. This set represents configurations where the robot may stop, so all the points in this set that are not in the target set must be completely avoided. The method we proposed in Chapter 2 was to compute the backprojection of the kernel considering $\mathbf{INT}(\hat{S}_{\mathbf{tc}}) \setminus \mathcal{T}$ as an artificial obstacle that must be completely avoided (no sliding on it is allowed).

The kernel computation presented in this chapter always creates sets $\mathbf{INT}(\hat{S}_{\mathbf{tc}})$ which are subsets of the target; therefore, no artificial obstacles are induced. However, this method does not necessarily yield the biggest possible backprojection. Indeed, only obstacles that lie between the backprojection and the kernel cause the backprojection to shrink. It is possible to grow the kernel without inducing any obstacles between the backprojection and the kernel. In this case, a bigger kernel produces a bigger backprojection.

Consider again our simple example. If we use a kernel where only the top edge of



(a) (b) (c)

Figure 3.14: Rectangular goal in free space

the rectangle is pushed inwards by 2ρ, the induced artificial obstacle (depicted in black in Figure 3.14(b)) leaves the top edge of the kernel free. Thus, for the commanded velocity shown in the figure, we get a bigger backprojection.

Yet, we can do better. Figure 3.14(c) shows that we can keep pushing the rays of the backprojection outwards, as long as the artificial obstacle does not interfere with the achievement of the kernel. The artificial obstacle starts creeping on the upper edge of the target set, and at some point it reaches the rays of the backprojection. At this point, the length of the segments of the two rays of the backprojection that lie in the target set is equal to 2ρ. Each ray can be moved outwards, up to the point where the vertex of the kernel (where this ray is incident) starts creating an artificial obstacle along the direction of this ray in front of the target set. Clearly, it is not possible to push the rays any further. Therefore, this is the maximal backprojection for the selected commanded velocity, i.e., the *preimage* of $\mathcal{T}$.

3.8.1 Maximal preimage of a convex set

The above method is applicable to arbitrary convex sets. Given a commanded velocity $\vec{\mathbf{v}}_d^c$ and a convex target set $\mathcal{T}$, the preimage of $\mathcal{T}$ is found by erecting two rays (a left and a right one) parallel to the edges of the inverted control uncertainty cone, such that they both cut segments of length 2ρ in the target set. The points where these rays exit $\mathcal{T}$ are called the *extreme points* of the kernel. The computation of the extreme points of the kernel can be done with two line-sweeps, one for each ray. During a sweep, we track the length of the segment that is the intersection of the sweep line and the target set. The orientation of the sweep line remains constant. We call the mapping of the sweep line location onto the length of its intersection with the target set the *chord function* $\kappa(\delta)$. The location of the line is represented by its distance δ from the point where the line is tangent to $\mathcal{T}$, with $\mathcal{T}$ being ahead of the line (hence, $\kappa(\delta) = 0$ when $\delta < 0$). When $\mathcal{T}$ is a convex set, $\kappa(\delta)$ consists of an increasing and a decreasing part and has a single global maximum. It is continuous except in the case where the sweep line is parallel to an edge of $\mathcal{T}$. The algorithm stops when the chord function becomes equal to 2ρ while increasing, or at $\delta = 0$ if $\kappa(0) \geq 2\rho$. The worst-case performance of this algorithm is $O(n \log n)$, where n is the complexity of $\mathcal{T}$.

In the absence of obstacles and other target sets, the two rays found with the above method constitute the upper boundary of the preimage $\mathcal{P}$. It remains to construct the kernel $\mathcal{K}$ that corresponds to $\mathcal{P}$. The kernel, which is part of the preimage, will define the lower boundary of $\mathcal{P}$. As mentioned before, no kernel point must be closer than 2ρ to any point in $\mathcal{P} \setminus \mathcal{T}$. Therefore, we can find the kernel by growing $\mathcal{P} \setminus \mathcal{T}$ by 2ρ and subtracting it from $\mathcal{T}$. In Subsection 3.5.3 we defined the notion of the cylsphere of an edge to be the set of points whose distance from the edge is less or equal to some number. Similarly, the cylsphere of any set of points is the set which contains all the points whose distance from the first set is smaller than a given distance:

$$\mathcal{K} = \mathcal{T} \setminus \mathit{cylsphere}(\mathcal{P} \setminus \mathcal{T}, 2\rho)$$

In most cases this computation can be performed by connecting the extreme points of the kernel with two lines, the *lower* and the *upper*. The lower line follows the

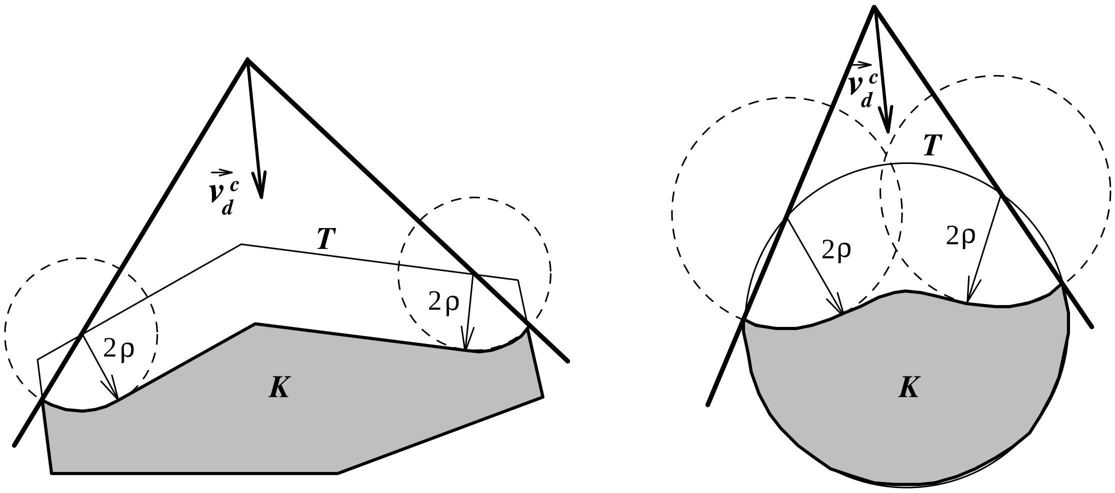


Figure 3.15: Maximal backprojections

lower boundary of the target set. The upper line is parallel to the upper boundary of the target set, at a distance 2ρ from it inwards. The upper line is connected to the extreme points of the kernel with circular arcs of radius 2ρ centered at the points where the rays enter the target set (see Figure 3.15). The termination condition is then found by growing the kernel isotropically by ρ.

Sometimes, though, the upper boundary may not lie entirely in the target set (see Figure 3.16). In this case, the above algorithm does not work. Then, we have to push the preimage rays inwards, until the part of it outside $\mathcal{T}$ and expanded by 2ρ is tangent to the lower boundary of $\mathcal{T}$. Thus, we obtain the widest possible preimage. The sweep-line algorithm used to find the original points can be adapted to handle this case as well.

3.8.2 About the existence of a unique maximal preimage

Another complication arises when the target consists of more than one convex sets. Consider the example of Figure 3.17(a). The target set consists of two parallelograms $\mathcal{T}_1$ and $\mathcal{T}_2$. If we use the above algorithm independently for each set, then the computed preimage may be incorrect. Assume, for example, that there exists a point A in the preimage of $\mathcal{T}_2$ and outside both $\mathcal{T}_1$ and $\mathcal{T}_2$, whose distance from the kernel of

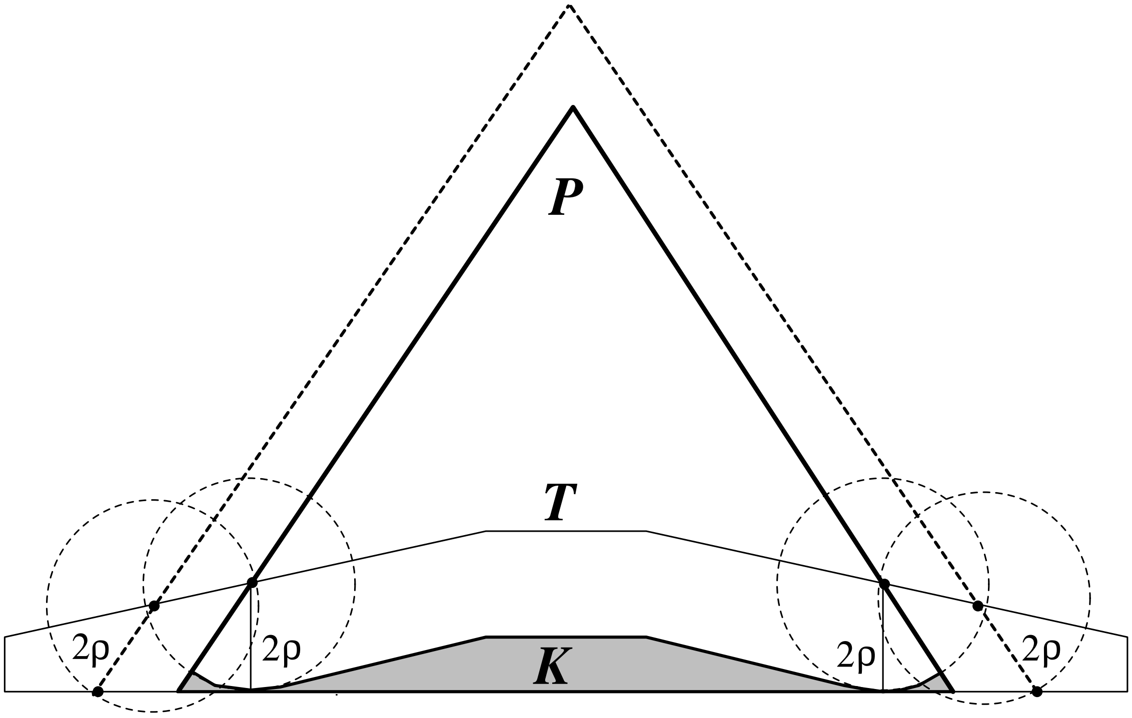


Figure 3.16: Preimage of a thin target region

$\mathcal{T}_1$ is less than 2ρ. A robot starting from A may actually think that it is at point B in the kernel of $\mathcal{T}_1$ and terminate its motion outside both target sets.

The problem is that the artificial obstacle created by the kernel of $\mathcal{T}_1$ intersects with the preimage of $\mathcal{T}_2$. When something like this occurs, there are two ways to proceed:

(a) We can leave the preimage of $\mathcal{T}_2$ intact and shrink the kernel of $\mathcal{T}_1$, so that their minimum distance is 2ρ. In practice, we can find the 2ρ-cylsphere of the preimage of $\mathcal{T}_2$, subtract it from $\mathcal{T}_1$, and then find the kernel and preimage of their difference. In the example of Figure 3.17(b), the kernel of $\mathcal{T}_1$ is split into two non-connected components, resulting in a much smaller preimage.[5]

(b) We can shrink the preimage of $\mathcal{T}_2$ up to the point that the upper boundary of the kernel of $\mathcal{T}_1$ remains intact (see Figure 3.17(c)). In this case, we find

---

[5] See the next subsection on how to find the preimage of a concave set.

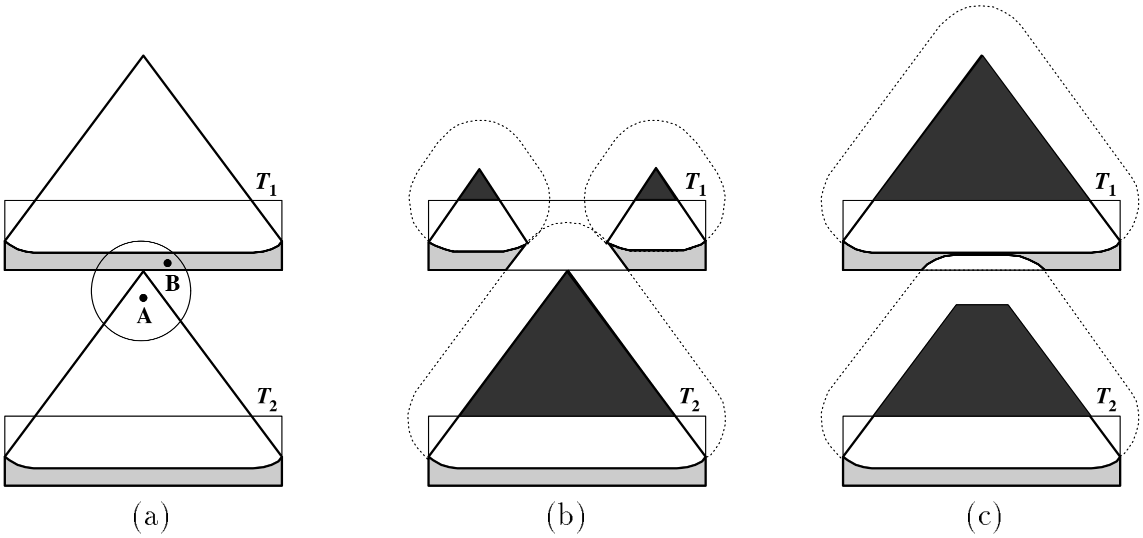


Figure 3.17: Preimage of non-connected convex sets

the 2ρ-cylsphere of the upper boundary of the kernel of $\mathcal{T}_1$ and subtract it from the preimage of $\mathcal{T}_2$.

We have thus computed two possible preimages of the target set $\mathcal{T}_1 \cup \mathcal{T}_2$. The union of these two preimages is not a preimage of the target set, because it contains points like A, where the robot may stop, although it is not in the target set. This means that there does not exist a unique two-dimensional set that contains all possible preimages of $\mathcal{T}_1 \cup \mathcal{T}_2$. In other words, we have proven that *there are cases when a unique maximal preimage*[6] *does not exist.* This result was first reported by Erdmann [32], who used non-compact sets to build a counterexample.

A complete algorithm must consider all possible maximal preimages, but it is very hard to do so because their number may be infinite (like in the example of Figure 3.17). One possibility is to compute the preimage independently for each set, and check for inclusion of all initial-region points. If this is true, there is hope that a guaranteed plan may exist. Then, we can find all points that are actually attainable by the robot during its motion (the forward projection), and check whether there is any chance for the robot to stop outside the goal. We can do this by computing the 2ρ-cylsphere of the part of the forward projection that lies outside the target set,

---

[6] A maximal preimage is a preimage that is not a proper subset of any other preimage.

and subtracting from the kernel any points that lie in this cylsphere. If the preimage of the reduced kernel still contains all initial-region points, then a guaranteed plan exists.

3.8.3 Maximal preimage of a concave set

The computation of the preimage of a concave set is significantly more complicated. The intersection of the sweep-line with the concave set may consist of more than one segments, so the chord function is a list of length functions $\kappa_i(\delta)$. These functions do not have the smooth behavior of the chord function of a convex set. They may have local maxima and minima, they may split into two new functions, or two of them may merge into one. Nevertheless, the sweep-line algorithm uses similar criteria to detect kernel extremities. It returns all points δ_0 such that $\kappa_i(\delta_0) = 2\rho$ while κ_i is increasing, $\delta_0 = 0$ if $\kappa_i(0) \geq 2\rho$, and all points δ_0 where two functions κ_i and κ_j merge, with $\lim\limits_{\delta \to \delta_0^-} \kappa_i(\delta) \leq 2\rho$, $\lim\limits_{\delta \to \delta_0^-} \kappa_j(\delta) \leq 2\rho$, and $\lim\limits_{\delta \to \delta_0^-} (\kappa_i(\delta) + \kappa_j(\delta)) \geq 2\rho$.

The above criteria may be satisfied more than once, so we may end up with multiple kernel extreme points and multiple preimage components. Furthermore, these components may interact with each other, in which case a unique maximal preimage does not exist. Figure 3.18 presents a complicated example with a concave target set. The kernel and the preimage consist of four disjoint components each, but no unique maximal preimage exists.

3.8.4 Recognition power

The termination conditions created with the above method, may ask the robot to stop, despite a sensory reading that is compatible with configurations outside the target set. They can do that because of the additional knowledge that is embedded in them, knowledge about the *initial state* and knowledge about the *final state*. By knowing the initial state (i.e., the preimage), the robot can rule out interpretations that are not achievable. By knowing the final state (i.e., the kernel) the robot can rule out interpretations that are beyond the kernel, since it is guaranteed that the robot will stop in the kernel.

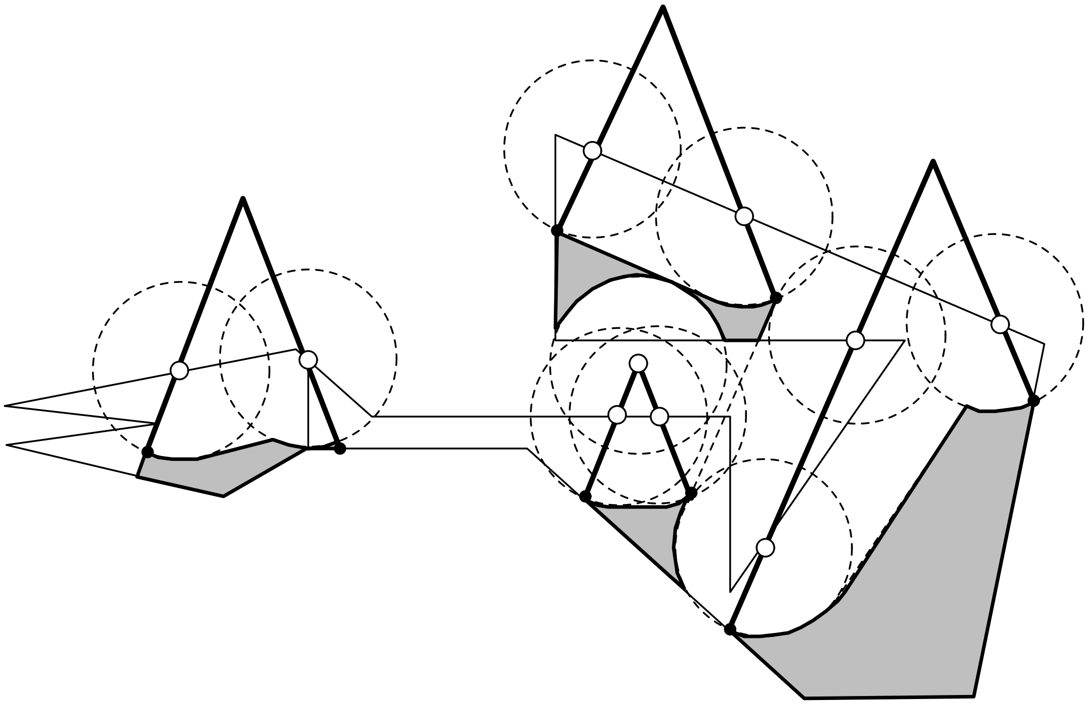


Figure 3.18: Maximal backprojection of a concave set

3.9 Omnidirectional backprojections

In the implemented planner of Section 3.7 the major bottleneck is the selection of the commanded velocity direction at each backchaining step. Not only do we lose completeness by discretizing the considered directions, but the resulting search takes exponential time in the number of backchaining steps.

The discretization problem can be solved if we find a way to efficiently compute for each point of the configuration space all commanded velocity directions that are guaranteed to bring a robot starting from that point into the target set $\mathcal{T}$.

Definition 3.1 (Nondirectional backprojection) *We call* nondirectional backprojection *of a target set $\mathcal{T}$ the two-dimensional set $\mathcal{NBP}(\mathcal{T}) \in \mathcal{C}$, consisting of all configurations, for which the above mapping produces a non-empty set of directions. In other words, $q \in \mathcal{NBP}(\mathcal{T}) \Leftrightarrow \exists d \in S^1 : \; q \in \mathcal{BP}(d; \mathcal{T})$.*

Definition 3.2 (Omnidirectional backprojection)
We call omnidirectional backprojection *of a target set* $\mathcal{T}$ *the three-dimensional set* $\mathcal{OBP}(\mathcal{T}) \in \mathcal{C} \times S^1$*, such that* $(q, d) \in \mathcal{OBP}(\mathcal{T}) \Leftrightarrow q \in \mathcal{BP}(d; \mathcal{T})$*.*

The knowledge of the nondirectional backprojection is sufficient to allow us to decide whether a one-step plan that connects a given initial region with a target set exists, but it does not facilitate the generation of the plan. In order to be able to derive the plan, we need to know the omnidirectional backprojection of the target set.[7] In the future, whenever we wish to refer specifically to the regular backprojection of a set, we will call it the *directional backprojection.*

The omnidirectional backprojection can be thought of as a stack of directional backprojections (*slices*) computed for various values of the commanded direction of motion. In order to create a one-step plan, we need to find out which of these slices completely include the initial region. This is the only reason we need the omnidirectional backprojection. Essentially, what we need is a one-dimensional boolean function that maps the various choices of the commanded direction of motion onto $[0, 1]$ (0 if the initial region is not included in the directional backprojection for this direction, 1 otherwise). It turns out that this function changes value only at a polynomial number of *critical orientations* of the commanded velocity vector. These values divide S^1 into regular sets (open intervals and points). If we compute the value of the boolean function for one point of such a set, we know it for all the other set points.

This observation was first made by Donald [25], though the notion of omnidirectional backprojections was introduced earlier in the original preimage backchaining paper [69]. The value of the boolean function can be computed by a sweep-line algorithm similar to the one that generates directional backprojections (Section 3.6). Moreover, this algorithm creates data structures that can be used to help update the value of the boolean function at each point considered, without having to call the sweep-line algorithm again. Donald proposed an $O(n^4 \log n)$ algorithm, with n being the number of obstacle edges, to compute nondirectional backprojections and

---

[7] There has been a fair amount of confusion between the notions of nondirectional and omnidirectional backprojection. Usually, the term nondirectional backprojection is used to describe both. We believe that the adjective "omnidirectional" is a better description of the three-dimensional set, since directional information is a vital part of it.

embedded this algorithm into a polynomial one-step planner. Briggs [8] reduced the time complexity of computing a nondirectional backprojection to $O(n^2 \log n)$.

Unfortunately, this algorithm cannot be generalized to produce multi-step plans. The reason is that as the backprojection varies with the choice of the commanded direction of motion, its kernel (the target set of the next backchaining step) varies too. Ultimately, of course, we are only interested in finding out whether a sequence of successive preimages includes the initial region or not. The answer to this question changes a large number of times (the number of changes is an exponential function of the number of backchaining steps), so it is not efficient to track it.

This result is typical of algorithms that use the preimage backchaining method. The core of our work, starting at the next chapter, shows how to avoid exponentiality by using carefully selected models of the robot, the workspace, and the representation of uncertainty.

Chapter 4

Landmark-Based Navigation

In this chapter we introduce the notion of *landmarks* as *islands of perfection* within an otherwise imperfect world. Using specific assumptions, we present a sound, complete and polynomial motion planning algorithm with uncertainty. The algorithm is based on the preimage backchaining method [69], and makes use of the idea of the omnidirectional preimage [25, 8]. We present an efficient way to compute and store a sufficient representation of an omnidirectional preimage, based on the idea of critical slices. Finally, we show several planning examples, and the simulated execution of the generated plans.

4.1 A new approach to motion planning with uncertainty

The problem of general motion planning with uncertainty is intrinsically hard. Several exponential lower bounds have been established, even for restricted subclasses of the problem. In particular, the three-dimensional problem attacked by the preimage backchaining method has been shown to be PSPACE-hard [77], as well as NEXPTIME-hard [15]. Canny gave a very general algorithm that takes double-exponential time in the number of plan steps (i.e., the number of motion commands in the plan) [14]. This algorithm assumes that the envelope of trajectories generated by the control system can be described algebraically, and that the robot has imperfect position and velocity sensing. The motion planning problem is reduced to the satisfiability of a semi-algebraic formula with alternating universal and existential quantifiers, like the one we introduced in Section 2.5. As Canny notes, "the existential quantifiers represent the actions taken by the plan executor, like choosing the direction to move or the time to stop moving, and the universals represent the actions of nature, namely the sensor readings." Its double-exponential bound stems precisely from the alternation of the quantifiers.

One exponential can be trimmed by eliminating sensing, so that the execution module does not need to make choices based on sensor readings. In this case, the number of alternations between universal and existential quantifiers is equal to the number of steps of the plan. Sensorless plans were first investigated in [37]. Donald [25] created a planner for a sensorless robot moving in a two-dimensional configuration space. The robot stops only by sticking on obstacle edges. Donald showed that this planner creates one-step motion plans in $O(n^4 \log n)$ time. Briggs [8] improved this bound to $O(n^2 \log n)$ using amortization techniques. Nevertheless, multi-step plans still require time exponential in the number of steps.

To make things worse, the number of steps for which a plan may exist grows with the complexity of the environment. Even in a three-dimensional polyhedral world, this number may grow exponentially with the environment complexity [15]. In order to avoid exponentiality, we need to find a way to limit the maximal number of steps of successful plans. Friedman [41] was able to do it by considering motions in the interior of a simple polygon, and assuming that sensing is possible only when the robot is in contact with one of the polygon edges. Under these assumptions, the maximum number of steps of a successful plan is bounded by the number of edges. Essentially, the polygon edges in Friedman's model are what in this work we call *landmarks*. Only by limiting sensing in a given number of regions in the workspace does it become possible to bound the number of planning steps, and thus to derive polynomial planning algorithms. This is exactly the key idea behind the work presented in this chapter. The framework, though, is significantly more general than Friedman's simple model.

Notions similar to landmarks have been introduced in the literature with a variety of names, e.g., *atomic regions* [12], *landmarks* [64], *signature neighborhoods* [70], *perceptual equivalence classes* [27], *sensory uncertainty field* [91], and *visual constraints* [49, 40]. Nevertheless, the relation between the use of landmarks and the development of complete polynomial planners has not been explored up to now. In this work we show that the use of landmarks considerably simplifies the selection of the set of states that the robot may traverse during plan execution, as well as the synthesis of state-recognition functions. It mainly reduces planning to selecting motion commands to navigate from landmark to landmark until the goal is achieved.

For a given problem, our planner generates a plan as a collection of motion commands, each attached to a specific region of the configuration space. When the robot enters one of these regions, it executes the associated motion command. Such a plan is reminiscent of the *reaction plans* proposed in [16, 28, 86] as a way to deal with uncertainty at planning time. In particular, Schoppers [86] developed the notion of *universal plans* whose rules cover all possible situations that may occur at execution time, much like our plans do. The interesting point is that our planner takes polynomial time, while the generation of universal plans is often believed to be exponential. However, unlike the planners mentioned above, our planner addresses a restricted family of planning problems in which uncertainty is well-bounded.

In this work we propose a new approach to robot motion planning with uncertainty. We focus on a restricted class of problems, for which we present a sound, complete, and polynomial planning algorithm. Then, we engineer a robot and its workspace so that these assumptions are met, and we perform experiments. Because the planner returns guaranteed plans, experimental failures can be tracked directly to violations of one or more of the assumptions. Thus, it is possible to isolate those assumptions that are the hardest (or the costliest) to satisfy, and then try to relax or eliminate them. The effect of the relaxation or elimination of a certain assumption on the complexity of the planning algorithm is an important part of our study. Since we do not wish to leave the realm of polynomial algorithms, we call this approach *the search for the limits of polynomiality in motion planning with uncertainty.*

4.2 Statement of the problem

We consider a circular omnidirectional robot moving in a planar workspace amongst circular forbidden regions that represent the obstacles. The definition of the obstacles as forbidden regions implies that the robot is not allowed to be (or move) in contact with them. Such a workspace maps onto a planar configuration space, where a point robot navigates amongst C-obstacles represented by forbidden circular regions (the *obstacle disks*). The configuration space also contains a set of *landmark disks* representing the regions of the configuration space, from where certain identifiable features of the workspace (the *landmarks*) are perceivable by the sensors of the robot.

The number of landmark disks is finite and equal to ℓ. The number of obstacle disks is also finite and in $O(\ell)$. (Throughout this chapter the number ℓ is used to measure the size of the input problem.) Landmark disks may intersect with each other; obstacle disks may intersect with each other, too. However, intersections between landmark and obstacle disks are not allowed. Maximal connected sets of landmark disks are called *landmark areas*, and their number is equal to s ($\leq \ell$). We assume that the robot has accurate knowledge of the configuration space model, i.e., it knows the exact location and size of the obstacle and landmark disks at planning time. This model does not change over time, i.e., landmarks and obstacles are stationary.

When the robot lies within a landmark disk, it can sense its configuration with perfect accuracy. Using its perfect sensing, the robot is able to navigate between any two points in the same landmark area without control error. This mode of navigation is called the *perfect-control mode.* When the robot lies outside all landmark disks, it has no sensing at all and it navigates using the *imperfect-control mode.*

A command in the perfect-control mode (called a *P-command*) is a sequence of regions $(R_1, R_2, \ldots, R_n)$ of the configuration space. The P-command simply prescribes that the robot should visit all these regions in order. We are not interested in the details of the execution of a P-command, as long as we know that it is guaranteed to succeed.

A motion command in the imperfect-control mode (an *I-command*) is described by a pair $(d, \mathcal{L}_T)$, where $d \in S^1$ is a direction in the plane, called the *commanded* *direction of motion*, and $\mathcal{L}_T$ is a set of landmark disks, called the *termination set* of the command. This command can be executed from anywhere in the configuration space 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 the *directional uncertainty cone*. The robot stops as soon as it enters a landmark disk in $\mathcal{L}_T$. 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 $\mathcal{I}$ (the *initial region*) that consists of one or several disks called the *initial-region disks*. The number of initial-region disks is assumed constant. 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 here. Thus, each initial-region disk may be disjoint from the landmark areas, or it may overlap with some of them, or it may be entirely contained in one of them. The robot must move into a given region $\mathcal{G}$, 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 motion commands in the perfect- and imperfect-control mode, which guarantees that the robot will be in $\mathcal{G}$ when the execution of the plan terminates.

The problem we have described above is a *planning problem*, and the model we have adopted is suitable for the purposes and capabilities of a planner. The model has to be correct (i.e., produce results that are compatible with the operation of a real-world system), but it does not have to describe every single capability of the system. This abstraction is exactly the focus of our research, which attempts to develop models simple enough for efficient planning, but in enough detail so that they are useful. The execution module need not be constrained to the use of our planning model; it may use its own (usually more precise) model to optimize execution efficiency. Nevertheless, both models must be able to recognize the language used to formulate plans.

4.3 Outline of a planning algorithm

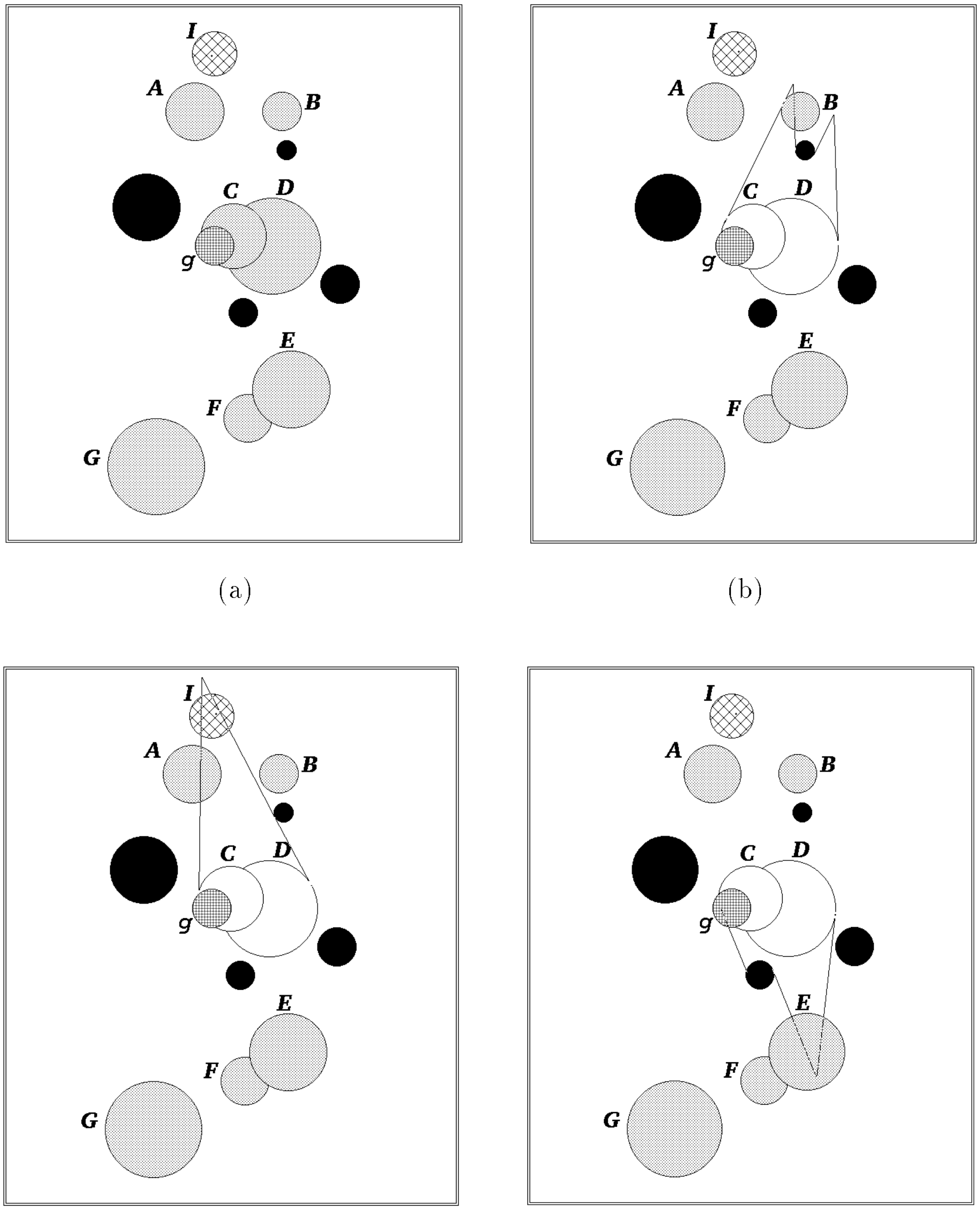
4.3.1 An example

Consider the example of Figure 4.1(a). The black disks represent obstacle disks and the grey disks represent landmark disks. There are four obstacle and seven landmark disks, but only five landmark areas. The initial region consists of a single disk marked with $\mathcal{I}$. The goal region consists of a single disk marked with $\mathcal{G}$.

The points of $\mathcal{G}$ that belong to some landmark disk are distinguishable from points outside the goal by means of the perfect position sensing provided by the landmark. Since $\mathcal{G}$ and landmark disk C have a common region, the robot can get from any point of C into the goal with the P-command $(\mathcal{G} \cap C)$. Similarly, the robot can get from D to C and from there to the goal with the P-command $(D \cap C, \mathcal{G} \cap C)$. But from no other landmark disk can the robot get into the goal using only the perfect-control mode. We say that C and D form the *extension set* $\mathcal{E}(\mathcal{G})$ of the goal. The remaining landmark disks are called the *intermediate goals*.

Since we cannot use the perfect-control mode to bring the robot into the extension set, we must use the imperfect-control mode. We are interested in guaranteed plans, so we need to find a commanded direction of motion for which, if the robot starts from any point in $\mathcal{I}$, it is guaranteed to get (and stop) in either C or D despite control error. This is equivalent to finding the preimage of the extension set for some commanded direction d (as defined in Chapter 3), and making sure that it contains $\mathcal{I}$. In this case, the computation of the preimage is not subject to the complications that arise from the interaction of goal reachability and recognizability. The reason is that landmark recognition occurs immediately after the robot enters a landmark region, and is independent of the way the landmark was achieved. Therefore, the preimage of a set of disks is equal to the backprojection of the set.

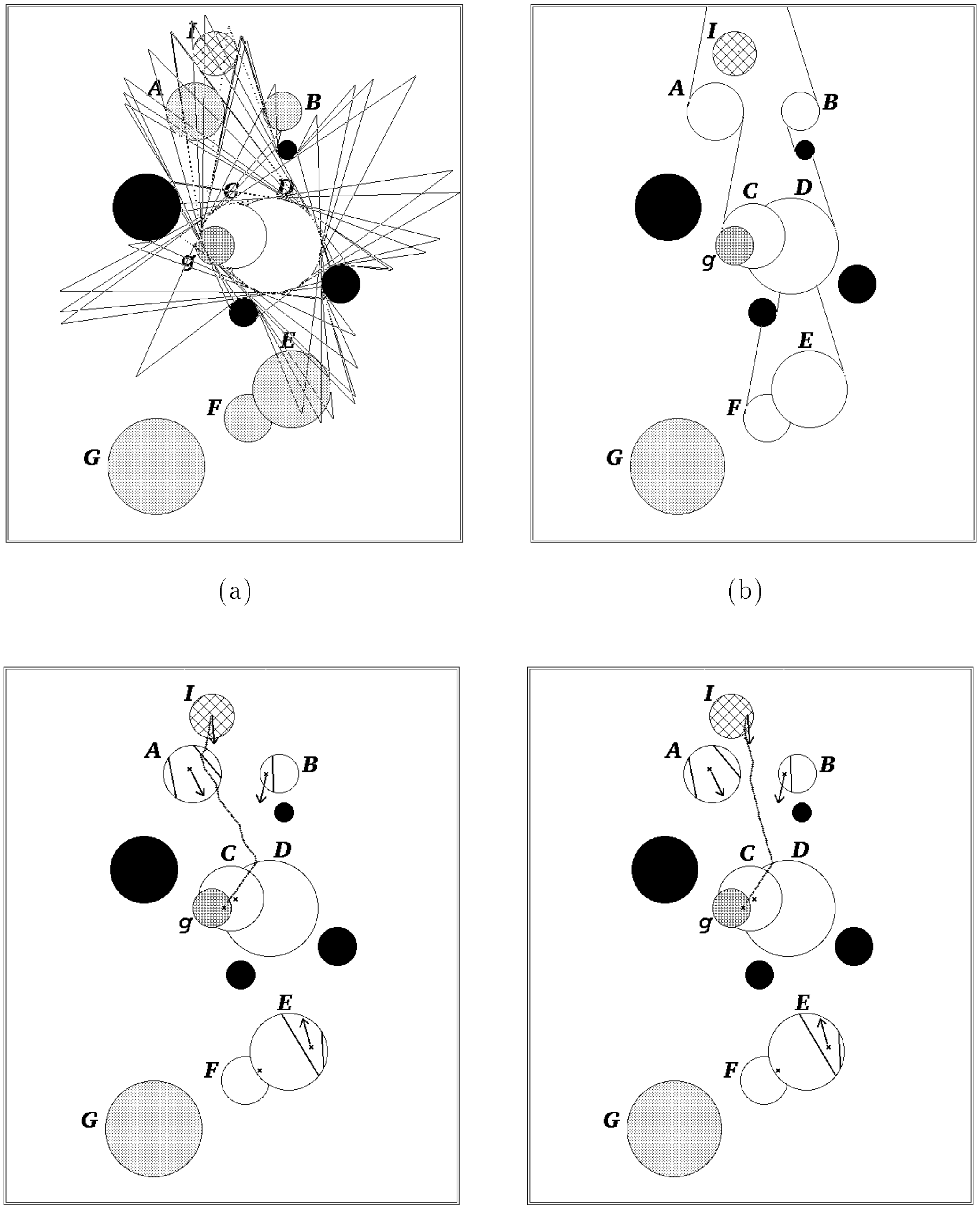
In Figure 4.2(a) the extension disks are white. Several preimages, computed for various values of the commanded direction d, are shown in the figure, but none of them completely includes the initial region $\mathcal{I}$. In fact, in this example there is no commanded direction d for which the preimage of C and D contains $\mathcal{I}$. Nevertheless,



(a) (b)

(c) (d)

Figure 4.1: An example



(a) (b)

(c) (d)

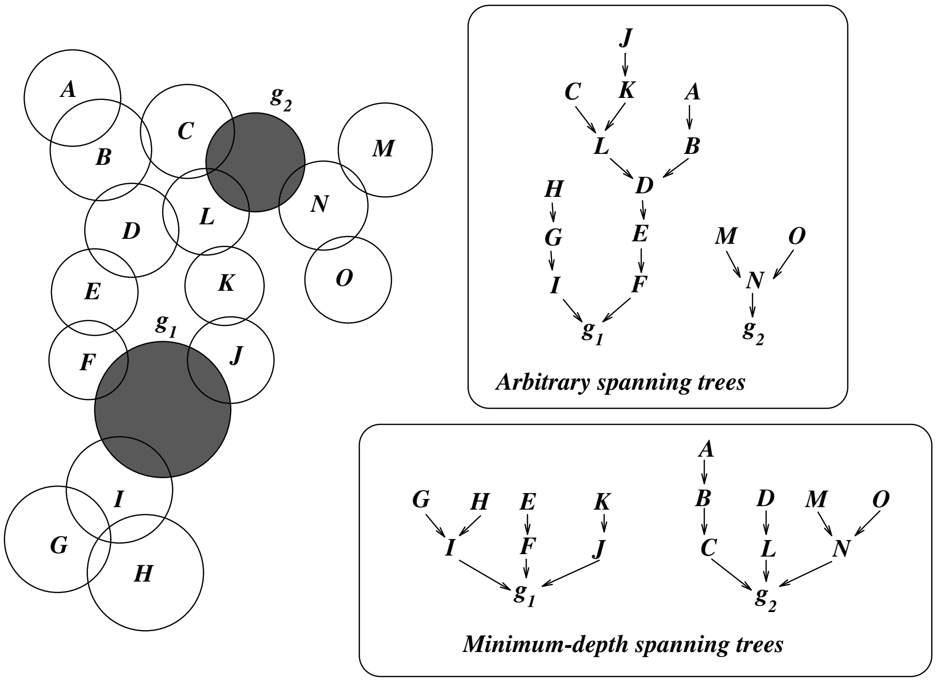
Figure 4.2: An example (cont.)

during our attempts we discover that intermediate-goal disks B, A, and E are intersected by some of the preimages (Figures 4.1 (b), (c), and (d)). From any point in these disks, as well as any other disks in the same landmark areas with them (disk F in this example), the robot can use the perfect-control mode to move into the intersection of a preimage with the disk, and then follow (in the imperfect-control mode) the commanded direction corresponding to this preimage in order to get into the extension set. From there, it can achieve the goal in the perfect-control mode.

Essentially, we have a guaranteed plan to lead the robot into the goal from any of these disks. So, we can add them to the extension set and perform the same procedure again. In Figure 4.2(b) we show this bigger extension and one preimage of it that contains $\mathcal{I}$. At this point we can stop and declare success. A guaranteed plan from $\mathcal{I}$ to $\mathcal{G}$ has been found.

This plan is shown in Figure 4.2(c). With the initial-region $\mathcal{I}$ we associate an I-command. The arrow attached to $\mathcal{I}$ represents the commanded direction of motion, and the termination condition is the achievement of any of the white disks (i.e., the disks in the final extension considered). To each white disk we attach one P-command and possibly one I-command. If an I-command is attached, it is represented by an arrow along the commanded direction of motion. The termination condition entails the achievement of any disk in the goal extension whose preimage intersected this disk when it was still an intermediate goal. In this case, the P-command corresponding to this disk is represented by an outlined subset of the disk containing the arrow (disks A, B, E in the figure). If no I-command is associated with a white disk, then that disk is necessarily intersecting another white disk or a goal disk (disks F, D, C). This intersection is the first region of the P-command attached to the disk.

In Figure 4.2(c) we also show a sample simulated execution of the plan. The robot starts from $\mathcal{I}$ and executes the associated I-command. When it hits A, it moves into the outlined area in the perfect-control mode, and then it executes the I-command attached to A. The termination condition of this command monitors the achievement of C or D. Indeed, when the robot hits D, it switches back to the perfect-control mode, moves into C, and from there into the goal. Another sample execution is depicted in Figure 4.2(d). In this case, the initial I-command brings the



(a) (b)

Figure 4.3: Creating P-commands

robot directly to D.

4.3.2 The method

If no point of the goal $\mathcal{G}$ lies in a landmark disk, then the goal is unrecognizable. Indeed, in this case no goal point can be distinguished from points outside the goal through sensing.[1] The planner returns `failure`, unless the initial region is a subset of the goal, in which case the *zero-step* plan consisting of the `NULL` command is guaranteed to leave the robot in the goal.

Otherwise, we identify all the landmark areas that intersect the goal, and we place all their disks in the initial goal extension set $\mathcal{E}(\mathcal{G})$. From each disk in the extension set we can get to the goal with a single P-command. The sequence of regions of such a P-command has the following characteristics: (a) Each region is the (non-null)

---

[1] Of course, if *all* points outside the landmark disks belong to the goal, then they are distinguishable from the points outside the goal. However, this is a rather unlikely scenario.

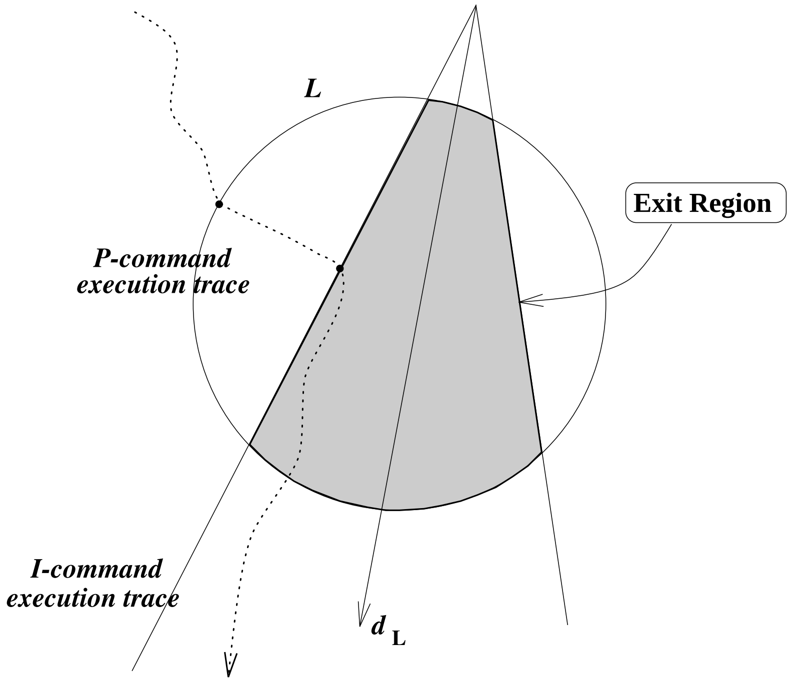


Figure 4.4: The exit region

intersection of two landmark disks, except the last region which is the (non-null) intersection of a landmark disk with a goal disk; (b) for each two consecutive regions there exists a landmark disk that contains both of them; and (c) there are no loops, i.e., the same region never appears twice in the sequence. When the robot enters an extension disk, it executes the P-command that is associated with the disk, and moves into the goal. One way to devise such P-commands is to establish a connectivity graph for all goal and extension disks and create spanning trees for the graph, with their roots being goal disks (as in the example of Figure 4.3). Since the spanning tree problem has several solutions, one may wish to select trees with minimum depth, thus minimizing at each extension disk the length of the P-command region sequence.

If the initial region is contained in the extension, then a plan has been found. Otherwise, we try to discover a commanded direction of motion $d_{\mathcal{I}}$, for which the preimage of the extension $\mathcal{P}(d_{\mathcal{I}}; \mathcal{E}(\mathcal{G}))$ includes the entire initial region. In such a case the I-command $(d_{\mathcal{I}}, \mathcal{E}(\mathcal{G}))$ is guaranteed to bring the robot into the extension set, from where it can move into the goal in the perfect-control mode. This is a *one-step* plan. In general, we consider one step to be the combination of an I-command and a P-command.

If no preimage contains the initial region, we identify all intermediate-goal disks (i.e., landmark disks that are not in the extension set) that are intersected by one of the preimages. Let L be such a disk and let $\mathcal{P}(d_L; \mathcal{E}(\mathcal{G})) \cap L \neq \emptyset$. The P-command $(\mathcal{P}(d_L; \mathcal{E}(\mathcal{G})) \cap L)$ and the I-command $(d_L, \mathcal{E}(\mathcal{G}))$ are attached to the disk (see the example in Figure 4.4). Furthermore, P-commands are attached to all disks in the same landmark area with L, as with the disks of the landmark areas that intersect the goal. This time, the spanning tree must have L as its root, and the last region of each P-command is precisely the intersection of L with the preimage. This intersection is called the *exit region*, because the robot has to get into it, before it can leave the landmark area of L. Thus, we start building a plan structure in a distributed fashion, by storing individual commands at each visited disk. At execution time, when the robot enters a disk that is part of the termination condition of the I-command that is currently being executed, the P-command associated with that disk is triggered. When the P-command terminates, the robot is either in the goal, or in a landmark disk with an associated I-command. In the first case, the execution of the plan halts; in the second case, the associated I-command is triggered.

From any point in the landmark areas that intersect a preimage of the extension set, the robot is guaranteed to get into the extension set in one step. Thus, we can add the disks in these landmark areas to the extension set and repeat this procedure recursively, until we either find a preimage that totally includes the initial region, or no more intermediate-goal disks can be intersected. In the first case, the algorithm returns `success`; in the second it returns `failure`. This recursive procedure is called *preimage backchaining*. Obviously, the number of backchaining steps cannot exceed the number s of landmark areas. Consequently, the number of steps in a successful plan is bounded by s. This is precisely the property that allows the planning algorithm to remain polynomial.

Lemma 4.1 *The number of steps in a successful plan is bounded by the number s of landmark areas.*

During execution, the robot moves from larger to smaller extension sets until it eventually reaches (and recognizes) the goal. The number of execution steps is bounded by the number of backchaining steps, but it is not necessarily equal to it (as in the example of Figure 4.2).

The following is pseudo-code for an algorithm that decides whether a plan exists or not. $\mathcal{I}$ is the initial region, $\mathcal{G}$ is the goal region, and $\mathcal{L}$ is the set of all landmark disks.

```
procedure $IsThereAPlan?(\mathcal{I}, \mathcal{G}, \mathcal{L})$ {
    if $(\mathcal{I} \subset \mathcal{G})$ return success;
    $E \leftarrow \mathcal{E}(\mathcal{G})$;
    if $(E = \emptyset)$ return failure;
    while $(\forall d \in S^1 \;\; \mathcal{I} \not\subset \mathcal{P}(d; E))$ {
        $IG \leftarrow \mathcal{L} \setminus E$;
        $D \leftarrow \emptyset$;
        $\forall d \in S^1 \; \forall g \in IG$ if $(\mathcal{P}(d; E) \cap g \neq \emptyset)$ then $D \leftarrow D \cup \{g\}$;
        if $(D = \emptyset)$ return failure;
    }
    return success;
}
```

4.3.3 Nondirectional and omnidirectional preimage

In Section 3.9 we defined the notions of directional, nondirectional and omnidirectional backprojections. These definitions can be readily extended to preimages.

Definition 4.2 (Nondirectional preimage) *The* nondirectional preimage *of a set $\mathcal{T}$ is the two-dimensional set $\mathcal{NP}(\mathcal{T}) \in \mathcal{C}$, consisting of all configurations q, for which there exists a commanded direction $d \in S^1$ such that $q \in \mathcal{P}(d; \mathcal{T})$.*

Definition 4.3 (Omnidirectional preimage) *The* omnidirectional preimage *of a set $\mathcal{T}$ is the three-dimensional set $\mathcal{OP}(\mathcal{T}) \in \mathcal{C} \times S^1$, such that $(q, d) \in \mathcal{OP}(\mathcal{T}) \Leftrightarrow q \in \mathcal{P}(d; \mathcal{T})$.*

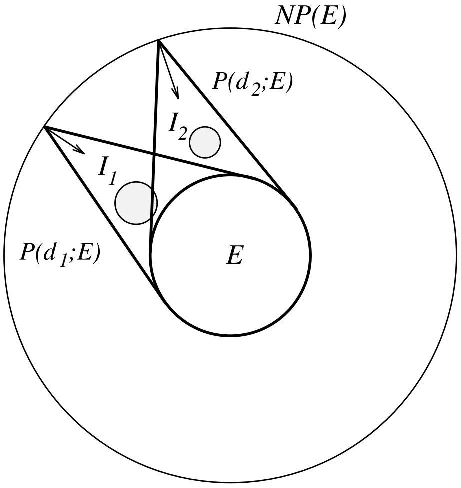


Figure 4.5: Nondirectional preimage

We use the term *directional preimage* to describe a regular planar preimage, when it is necessary to explicitly differentiate it from nondirectional and omnidirectional preimages.

Note that covering the initial region with the nondirectional preimage is not sufficient to guarantee the existence of a plan. Consider the example of Figure 4.5. There is one extension disk E and two initial-region disks I_1 and I_2. The nondirectional preimage of E is a bigger disk $\mathcal{NP}(E)$, which covers both initial-region disks. However, no directional preimage of E covers both disks. If $\mathcal{P}(d_1; E)$ covers I_1 and $\mathcal{P}(d_2; E)$ covers I_2, then the robot has to know from which initial-region disk it starts, in order to decide whether to follow d_1 or d_2. However, if both I_1 and I_2 lie outside all landmark disks, the robot may not be able to know from which region it starts.

In general, nondirectional preimages are not useful for planning, because they do not provide directional information. It is the omnidirectional preimages that contain this information, so we focus our attention on them. The omnidirectional preimage has been defined as a three-dimensional set, however, because of the constraint that the robot moves with constant commanded velocity direction (which can only change when the robot is not moving), we can represent the omnidirectional preimage as the *disjoint union*[2] of all directional preimages for all possible values of $d \in S^1$. Thus, we

---

[2] as opposed to the regular union used to compute nondirectional preimages

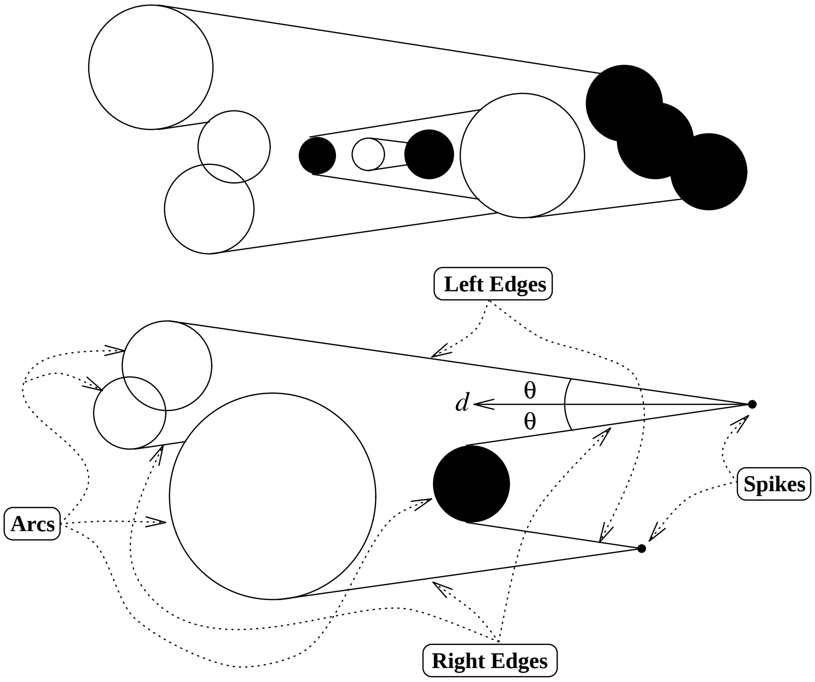


Figure 4.6: A preimage

represent an omnidirectional preimage of a set of configuration space points $\mathcal{E}$ with the set $\{(d, \mathcal{P}(d;\mathcal{E})) \mid d \in S^1\}$. Each member of this set is called a *slice*. The robot can only move on a particular slice, or vertically across slices. It turns out that it is not necessary to compute all possible slices (which are infinite); for the purposes of the planning algorithm, the omnidirectional preimage can be sufficiently represented by a polynomial number of slices, for appropriately selected values of d.

4.4 Computation of directional preimages

4.4.1 Description

A directional preimage may consist of one or more connected components. Each component may have holes, which in turn may contain other components of the preimage. In Figure 4.6 we show a sample preimage with three components. One of them lies in the hole of another component. The boundary of a component consists of circular segments called *arcs* and straight segments called *edges*. Each arc is a subset of the boundary of an extension or an obstacle disk. The *right ray* (resp. *left ray*) of an extension disk is the half-line tangent to the disk, erected from the tangency point in the direction pointed by $\pi + d + \theta$ (resp. $\pi + d - \theta$) that leaves the disk to its left (resp. right). The same definition holds for obstacle disks, only in this case the obstacle disk lies to the right of a right ray and to the left of a left ray. Each edge is contained in the right or left ray of some extension or obstacle 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 extension or obstacle disk or another erected ray. The right (or left) ray of any 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*. A component may have zero or more spikes, but never more than one plus the number of obstacle disks that form part of the component's boundary. In the example of Figure 4.6, the upper two components have no spikes, whereas the lower component has two.

The description of a preimage is a collection of circular lists consisting of arcs and rays. Each list corresponds to a contiguous outer or hole boundary of a component.[3] The total length of these lists is a linear function of the number of extension and obstacle disks. Indeed, it is known that the boundary of the union of ℓ disks is linear in ℓ [46]. Thus, the number of arcs in the boundary of a preimage, whose endpoints do not coincide with the origin or endpoint of an edge is linear in ℓ. Now, each ray supports at most one preimage edge. There are only two rays per disk, so the total number of edges is also linear in ℓ. Finally, if an endpoint of an arc coincides with the origin or the endpoint of an edge, then that arc can be attributed to that particular edge. Obviously, only two arcs can be attributed to a single edge, so the total number of arcs remains linear in ℓ. Thus, we have proved the following lemma:

Lemma 4.4 *The description of a preimage has size $O(\ell)$.*

---

[3] In the following, there is no need to differentiate between an outer and a hole boundary list.

4.4.2 Computation

Before we compute the preimage we need to precompute all landmark areas and all obstacle areas.[4] This precomputation is carried out once, using a divide-and-conquer algorithm that takes $O(\ell \log^2 \ell)$ time [82]. The algorithm identifies all disks belonging to a particular area and constructs its boundary as one or several lists of circular arcs. At the same time, it finds out which initial-region disks are fully contained in a landmark area, and whether a landmark area intersects the goal or not.

At this point we have a linear description of the boundary of the extension set. Using this description and a sweep-line algorithm, we can compute the preimage of the extension set in time $O(\ell \log \ell)$. The sweep line is perpendicular to the commanded velocity of motion d, and moves in a direction opposite to d. The event queue contains *x-points* corresponding (a) to the sweep line being tangent to an extension landmark area or an obstacle area (*start-* and *end-points*), (b) to vertices of the extension landmark areas and the obstacle areas (*vertex-points*), (c) to a left or right ray being tangent to an extension landmark area or an obstacle area (*l-tangent-* and *r-tangent-points*), and (d) to intersections of rays with the boundary of an extension landmark area or an obstacle area (*intersection-points*, only one per ray). The total number of x-points in the queue is linear in ℓ.

Lemma 4.5 *The computation of a preimage takes $O(\ell \log \ell)$ time.*

With the addition of a few additional x-points it is possible to find out in the same time bound which initial-region disks are included in the preimage, and which intermediate-goal areas are intersected by the preimage. These x-points correspond (a) to the sweep line being tangent to initial-region disks and intermediate-goal areas, (b) to intersections of the boundary of the preimage with initial-region disks or the boundaries of intermediate-goal areas, and (c) to intersections of the initial-region disks among themselves and with the boundary of intermediate-goal areas.

---

[4] An obstacle area is defined similarly to a landmark area as a maximal connected set of obstacle disks.

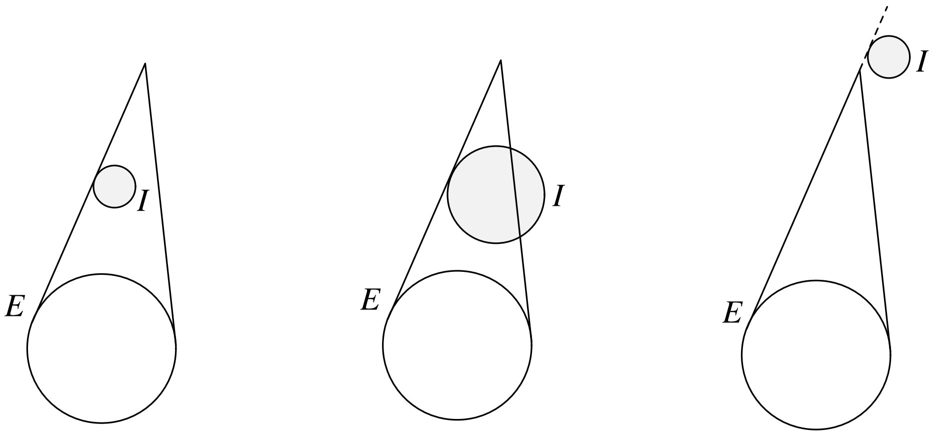
4.5 Computation of omnidirectional preimages

In the previous section we described how to compute the preimage $\mathcal{P}(d; \mathcal{E})$ of a set of extension disks $\mathcal{E}$ for a given commanded direction of motion d. Theoretically, in order to compute the omnidirectional preimage, we need to call the sweep-line algorithm for all possible values of d, i.e., an infinite number of times. Fortunately, this is not necessary. The planning algorithm does not really require the computation of the omnidirectional preimage. It only requires answering the following questions for all possible values of $d \in S^1$:

- **Inclusion:** Is the initial region included in the preimage?
- **Intersection:** Which intermediate-goal disks are intersected by the preimage?

We call these questions the *significant questions* for planning. As d varies from 0 to 2π, the answers to the significant questions do not change continuously, but only at specific values of d, called *critical orientations*. Whenever the answer to one of the significant questions changes, we say that a *critical event* has occurred. The answers remain unchanged throughout the interval between any two successive critical events. Thus, it is only necessary to compute these functions once per such interval. This computation can be performed with the sweep-line algorithm described in the previous section.

Consider the very simple example of Figure 4.7, where there is only one extension disk E and only one initial-region disk I. The answer to the question whether I belongs to $\mathcal{P}(d; E)$ can change from `NO` to `YES` only when the left ray stemming from E becomes tangent to I, with I being to the right of the ray (Figure 4.7(a)). However, this may not happen at this orientation, either because I is too large (Figure 4.7(b)), or because it is too far away (Figure 4.7(c)). Consequently, we cannot know whether this orientation is critical or not without computing the preimage. We say that this orientation is *potentially critical*. Similarly, there exists another potentially critical orientation, when the right ray becomes tangent to I, with I being to the left of the ray. In this case the answer may change from `YES` to `NO`. These are the only potentially critical orientations in this example. They divide S^1 in two intervals, so we need to



(a) (b) (c)

Figure 4.7: Potentially critical event

run the sweep-line algorithm only twice.

Furthermore, if at the moment when the left ray becomes tangent to I we have a description of the preimage, then it is possible to check whether a critical event will occur or not, without computing the preimage. Indeed, we can compute the length of the left edge at this orientation and make sure that it is greater than the length of the external common tangent of the two disks. If, additionally, no other edges or arcs of the preimage boundary intersect I at this orientation, then we can be sure that a critical event will occur, and consequently I will become fully contained in the preimage. In this case, there is no need to call the sweep-line algorithm. This reasoning requires that we keep track of the topological description of the preimage. Once again, this description does not change continuously, but only at some additional critical orientations.

All critical events may occur only when a ray of the preimage becomes tangent to a disk, or a spike of the preimage intersects a disk. We differentiate the various kinds of critical events by naming them as follows: When the topological description of the preimage changes, we have a *T-critical* event. T-critical events involving extension disks are called *E-critical*; those involving obstacle disks are called *O-critical*. When the inclusion of the initial region changes, we have an *I-critical* event. Finally, when the set of intermediate-goal disks that are being intersected by the preimage changes, we have an *L-critical* event.[5]

Potentially critical orientations can be precomputed ahead of time, then sorted in counterclockwise order and stored in a queue. The description of the preimage and the answers to the significant questions can be computed for an arbitrary direction d_0 with one run of the sweep-line algorithm. Then, we can start popping potentially critical orientations from the queue, starting with the first one after d_0, thus performing an *angular sweep*. For each such orientation we check whether it is truly critical (using the topological description of the preimage we have), and, if it is, we update the description of the preimage and/or the answers to the significant questions. We continue until we have processed all potentially critical orientations in the queue, or until we discover that all initial-region disks are included in the preimage for some value of d, in which case we declare success and return a plan. During the angular sweep, we keep track of the intermediate-goal disks that have been intersected by the preimage for various values of d, and for each such disk we record a description of its intersection with one of the preimages along with the corresponding value of d. This information is used to create the motion commands to be attached to each intersected intermediate goal: The intersection of a preimage with the disk is the only region of the P-command attached to the disk, and the recorded value of d is the commanded direction of the disk's I-command. The termination condition of the I-command monitors the achievement of one of the disks in the current extension set.

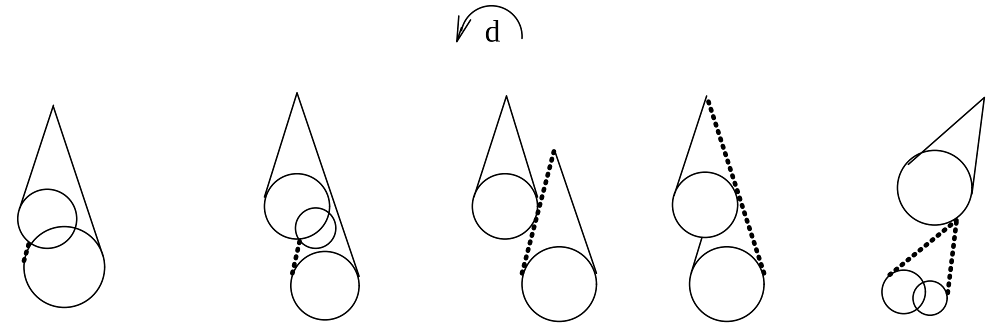
If no plan has been found after one angular sweep all intersected intermediate goals are inserted in the extension set, and the algorithm is called recursively according to the preimage backchaining method.

4.5.1 Potentially critical events

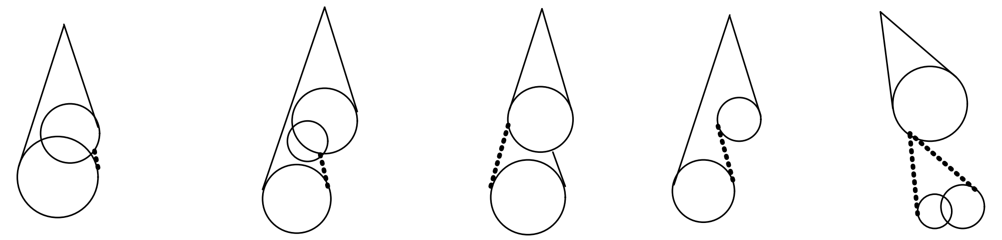
We now give a complete list of potentially critical events. In all the figures, extension disks are white, obstacle disks are black, intermediate-goal disks are grey and initial-region disks are light grey. The classification of events assumes counterclockwise movement of the commanded velocity orientation d.

---

[5] "L" stands for "landmark"



(a) E-Left-Birth (c) E-Left-Vertex (e) E-Reach (g) E-Leave (i) E-Spike-In



(b) E-Right-Death (d) E-Right-Vertex (f) E-Include (h) E-Exclude (j) E-Spike-Out

Figure 4.8: E-critical potential orientations

■ **E-critical events** (see Figure 4.8)

- *E-Left-Birth:* A new left edge emerges at the intersection of two extension disks.
- *E-Right-Death:* A right edge disappears at the intersection of two extension disks.
- *E-Left-Vertex:* The endpoint of a left edge crosses the intersection between two extension disks.
- *E-Right-Vertex:* The endpoint of a right edge crosses the intersection between two extension disks.
- *E-Reach:* A left edge reaches an extension disk by becoming tangent to it.
- *E-Include:* A left edge leaves the extension disk that contains its endpoint by becoming tangent to it.
- *E-Leave:* A right edge reaches an extension disk by becoming tangent to it.
- *E-Exclude:* A right edge leaves the extension disk containing its endpoint by becoming tangent to it.
- *E-Spike-In:* A spike reaches an extension disk.
- *E-Spike-Out:* A spike leaves an extension disk.

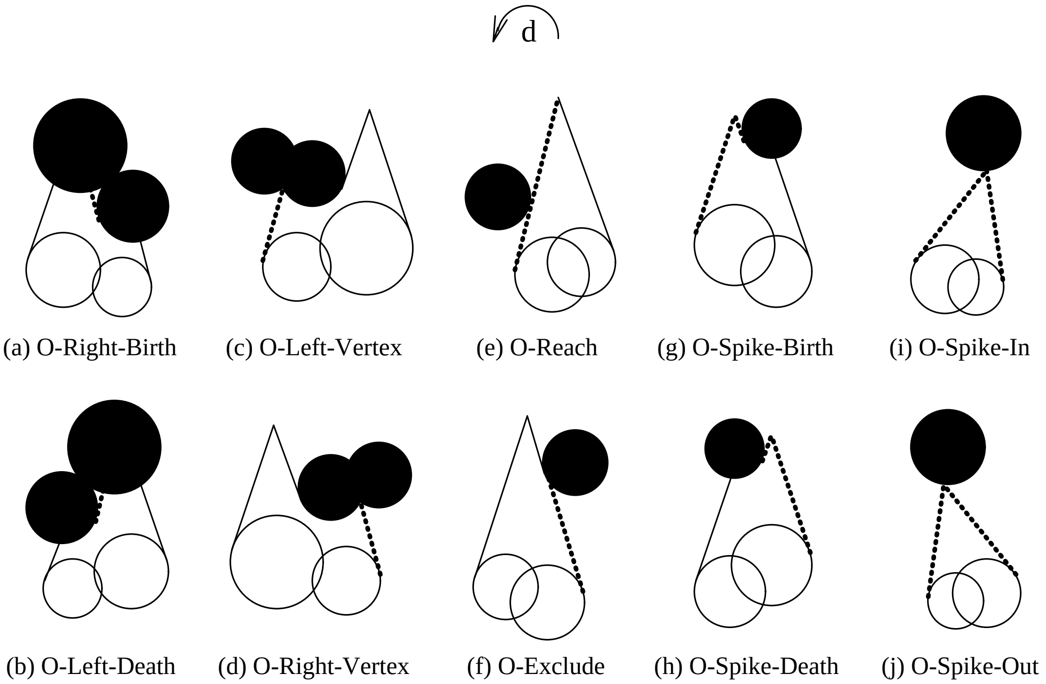


Figure 4.9: O-critical potential orientations

■ **O-critical events** (see Figure 4.9)

- *O-Right-Birth:* A right edge emerges at the intersection of two obstacle disks.
- *O-Left-Death:* A left edge disappears at the intersection of two obstacle disks.
- *O-Left-Vertex:* The endpoint of a left edge reaches the intersection of two obstacle disks.
- *O-Right-Vertex:* The endpoint of a right edge reaches the intersection of two obstacle disks.
- *O-Reach:* A left edge reaches an obstacle disk by becoming tangent to it.
- *O-Exclude:* A right edge leaves an obstacle disk by becoming tangent to it.
- *O-Spike-Birth:* A spike emerges as a left edge terminating on an obstacle disk reaches the point where a right edge arises from this disk.
- *O-Spike-Death:* A spike vanishes as its left edge, pushed by its right edge, shortens to zero length against an obstacle disk.
- *O-Spike-In:* A spike hits an obstacle disk and disappears.
- *O-Spike-Out:* A spike appears on the boundary of an obstacle disk.

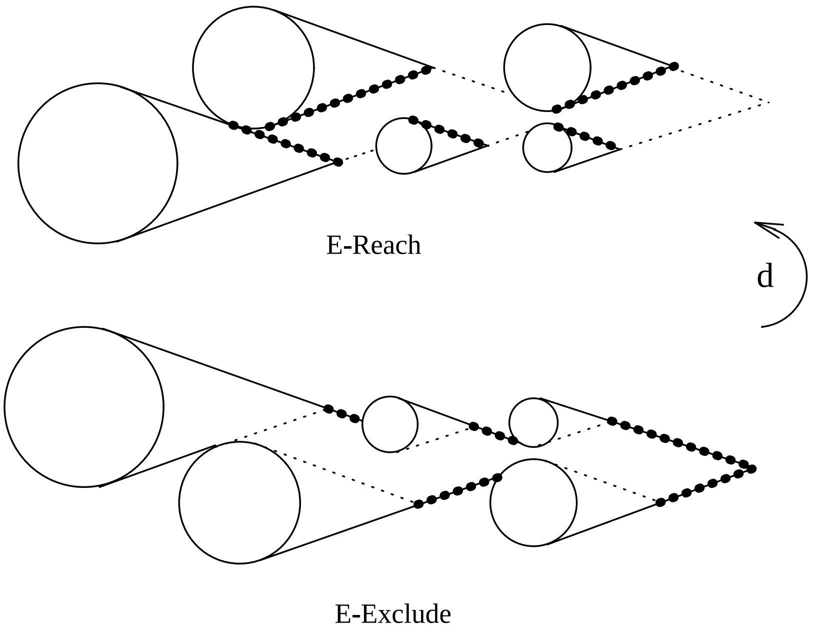
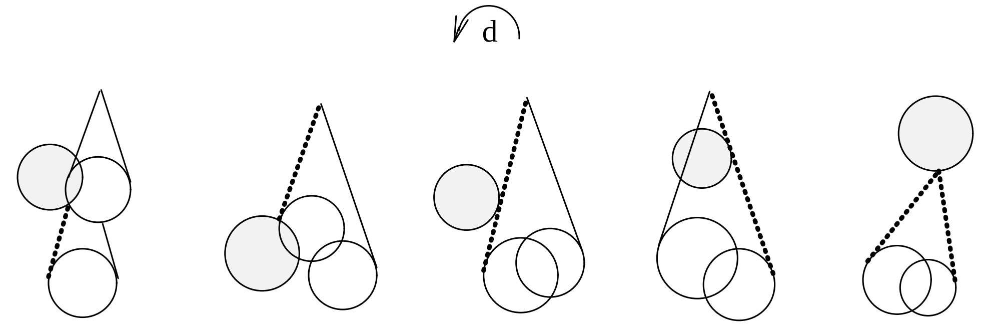


Figure 4.10: Catastrophic events

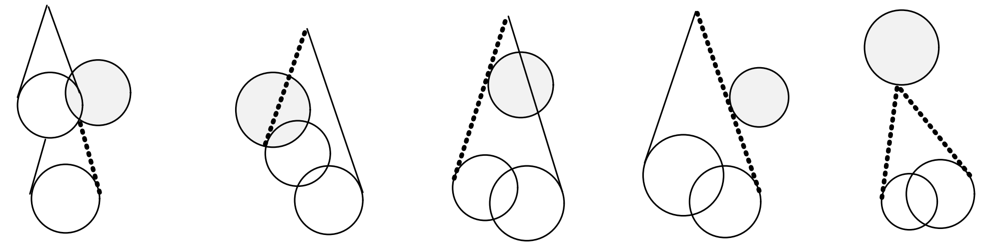
There are $O(\ell)$ extension disks and obstacles. Each disk can produce at most two rays, so the number of rays is also $O(\ell)$. As mentioned before, the size of the boundary of the union of $O(\ell)$ disks is also in $O(\ell)$. Hence, there are $O(\ell)$ E-Left-Birth, E-Right-Death, O-Right-Birth and O-Left-Death events, and $O(\ell^2)$ E-Left-Vertex, E-Right-Vertex, O-Left-Vertex and O-Right-Vertex events. The E-Reach, E-Include, E-Leave, E-Exclude, O-Reach, O-Include, O-Leave and O-Exclude events occur when one of the $O(\ell)$ rays becomes tangent to one of the $O(\ell)$ extension or obstacle disks, so their number is in $O(\ell^2)$. Finally, spike events combine two distinct rays and a disk, so their number is $O(\ell^3)$.

Lemma 4.6 *There are $O(\ell^3)$ T-critical orientations.*

Most T-critical events cause only a local change in the description of the preimage, so they can be processed in constant or logarithmic time. Instead of recomputing the preimage from scratch, we just update its description. However, an E-Reach event and an E-Exclude event may cause widespread changes to the preimage (See Figure 4.10). The number of these changes may be linear in the number of extension and obstacle



(a) I-Left-Vertex (c) I-Vertex-Cross (e) I-Reach (g) I-Leave (i) I-Spike-In



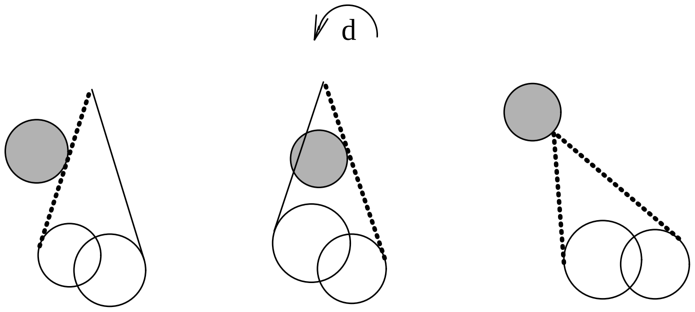
(b) I-Right-Vertex (d) I-Vertex-Cross (f) I-Include (h) I-Exclude (j) I-Spike-Out

Figure 4.11: I-critical potential orientations

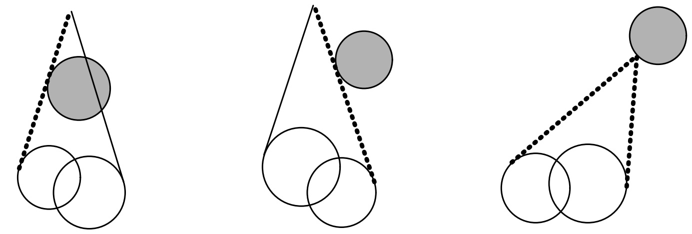
disks. These events are called *catastrophic*. At each catastrophic event we recompute the preimage from scratch using the line-sweep algorithm. Catastrophic events may cause the answers of the significant questions to change as well, so according to our definition they are potentially I-critical and L-critical too. Nevertheless, since we have already considered them, we omit them from the following lists of I-critical and L-critical events.

■ **I-critical events** (see Figure 4.11)

- *I-Left-Vertex:* A left edge crosses the intersection of the extension disk containing its endpoint with an initial-region disk.
- *I-Right-Vertex:* A right edge crosses the intersection of the extension disk containing its endpoint with an initial-region disk.
- *I-Vertex-Cross:* The origin of an edge crosses the intersection of its extension disk with an initial-region disk.
- *I-Reach:* A left edge reaches an initial-region disk by becoming tangent to it.



(a) L-Reach (c) L-Leave (e) L-Spike-In



(b) L-Include (d) L-Exclude (f) L-Spike-Out

Figure 4.12: L-critical potential orientations

- *I-Include:* A left edge leaves an initial-region disk by becoming tangent to it.
- *I-Leave:* A right edge reaches an initial-region disk by becoming tangent to it.
- *I-Exclude:* A right edge leaves an initial-region disk by becoming tangent to it.
- *I-Spike-In:* A spike enters an initial-region disk.
- *I-Spike-Out:* A spike exits 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.

Lemma 4.7 *There are $O(\ell^2)$ I-critical directions.*

■ **L-critical events** (see Figure 4.12)

- *L-Reach:* A left edge reaches an intermediate-goal disk by becoming tangent to it.
- *L-Include:* A left edge leaves an intermediate-goal disk by becoming tangent to it.

- *L-Leave:* A right edge reaches an intermediate-goal disk by becoming tangent to it.
- *L-Exclude:* A right edge leaves an intermediate-goal disk by becoming tangent to it.
- *L-Spike-In:* A spike enters an intermediate-goal disk.
- *L-Spike-Out:* A spike exits an intermediate-goal 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.

Lemma 4.8 *There are $O(\ell^3)$ L-critical directions.*

4.5.2 Technical details of the angular sweep algorithm

Up to now we have given a rather vague description of the angular sweep algorithm. We now present it in more detail.

First, the line-sweep algorithm is called for an arbitrary direction d_0 and returns the following information:

- A topological description of the preimage, consisting of circular lists of arcs and edges. Each list represents a maximal contiguous (hence, closed-loop) boundary of the preimage.
- For each initial-region disk, the number of intersections with the boundary of the preimage. This number is called the *incount* of the disk. If there are no intersections, then the disk is either completely outside the preimage ($incount = 0$), or it is contained in it. In this last case, the sweep-line algorithm returns $incount = -1$.
- For each intermediate-goal disk, the disjoint regions that form the intersection of the disk with the preimage. The number of these regions may be zero, one, or any number less than the total number of landmarks and obstacles.

We now proceed with the angular sweep. All angular critical events are caused by rotating edges and spikes. They are being kept in a queue, sorted in counterclockwise order starting from the arbitrary initial orientation d_0. Although it is possible to schedule all potentially critical events ahead of time, it is not necessary to do so.

Since we already have a description of the preimage, we know which edges and spikes actually exist, so we can schedule only those events caused by existing edges and spikes. The rays supporting these edges, and the pairs of rays forming these spikes are marked so that they will not be reprocessed in the future. Thus, the same event is never scheduled twice.

Rays cause critical events when they become tangent to another disk. It is straightforward to compute the value of d for which this happens. Spikes cause critical events when they intersect other disks. If the two rays forming the spike stem from the same disk, then the locus of the spike as d moves counterclockwise is a circular arc. Its intersections with other disks define the spike events. However, if the two rays of the spike stem from different disks, then the locus of the spike is a fourth degree curve (discussed in Appendix A). Fortunately, its intersections with other disks are the roots of a fourth degree polynomial and they can be found algebraically.

When the scheduling phase is over, we start popping events from the queue. For each event we verify that it is an actual critical event, and if so, we process it. Processing depends on the type of the event.

At T-critical events, the topological description of the preimage changes. If new (unprocessed) rays or spikes appear, the critical events they may cause as the angular sweep proceeds are inserted in the queue. At catastrophic events, the *incount* of initial-region disks as well as the intersection of intermediate-goal disks with the preimage may change. In this case, we obtain the updated values with the same call to the sweep-line algorithm used to compute the new description of the preimage.

At I-critical events, the *incount* of some initial-region disk changes. If before the event *incount* is -1 or 0, then necessarily after the event *incount* $= 2$. If, on the other hand, *incount* is 2 before the event and goes to 0 after the event, we have to check whether the disk becomes included in the preimage, or moves completely outside it. This can be decided based on the type of event that occurs. For example, if the event is an I-Include event, then the disk lies in the preimage and we set *incount* $= -1$; if the event is an I-exclude event, then the disk is outside the preimage and we let *incount* be 0.

At L-critical events, a new region is added to the intersection list of an intermediate-goal disk, or a region is deleted, or the topological description of a region changes. After the angular sweep is over, each intersected intermediate-goal disk has a list of regions (topological descriptions, along with the values of d for which they are valid) associated with it. In order to create the motion commands associated with the disk, we must select one of the regions and one of the associated values of d. In our implementation, we perform this selection attempting to maximize the intersection region. Although region selection is irrelevant under the assumptions of this chapter, it becomes important when we start considering some uncertainty in landmark areas.

4.6 The complexity of the planning algorithm

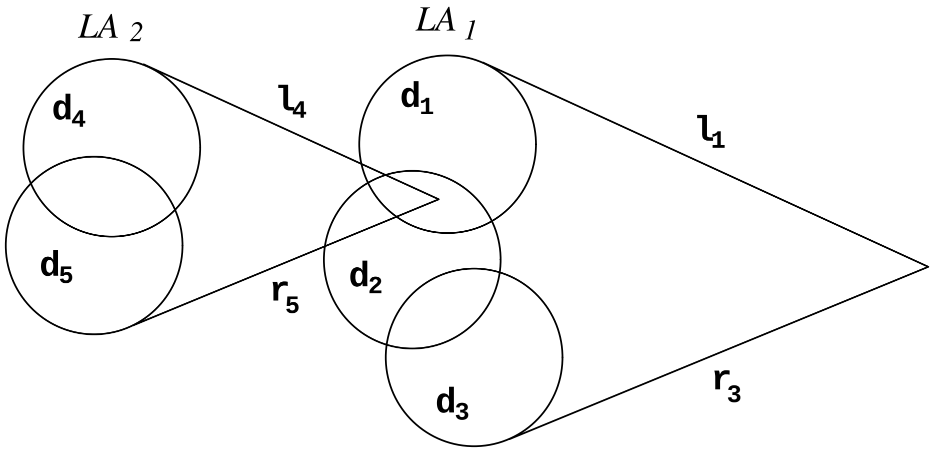
The number of potentially critical events is in $O(\ell^3)$. Scheduling them in a height-balanced tree (AVL-tree) takes $O(\ell^3 \log \ell)$ time. Processing of most events can be done in constant or logarithmic time, thus it requires time $O(\ell^3 \log \ell)$. However, catastrophic events require running the line-sweep algorithm that takes $O(\ell \log \ell)$. Fortunately, there are only $O(\ell^2)$ catastrophic events, so their total contribution to the complexity is $O(\ell^3 \log \ell)$. Overall, one step of the backchaining process takes time $O(\ell^3 \log \ell)$. Since we have proved that there can be at most s (the number of landmark areas) backchaining steps, we deduce:

Lemma 4.9 *The complexity of the planning algorithm is* $O(s\ell^3 \log \ell)$.

This bound may not be a tight estimate of the complexity of the algorithm. In practice, the algorithm seems to run much faster than this estimate. This is an indication that there may exist a stricter lower bound. The major contributors to the asymptotic complexity bound are the following:

- The E-, O- and L-Spike events, whose number is $O(\ell^3)$.
- The call to the line-sweep algorithm at catastrophic events.

The contribution of L-Spike-In and L-Spike-Out events can be easily discounted. Indeed, if an intermediate goal is hit by a spike, then at the next backchaining step



$$(\underbrace{\texttt{l}_1}_{\alpha}\ \texttt{d}_1\ \texttt{l}_4\ \underbrace{\texttt{d}_4\ \texttt{d}_5}_{\beta}\ \texttt{r}_5\ \texttt{d}_2\ \underbrace{\texttt{d}_3\ \texttt{r}_3}_{\gamma}) \longrightarrow \begin{cases} (\texttt{l}_1\ \texttt{d}_1\ \texttt{d}_2\ \texttt{d}_3\ \texttt{r}_3) \\ (\texttt{l}_4\ \texttt{d}_4\ \texttt{d}_5\ \texttt{r}_5) \end{cases}$$

Figure 4.13: Hidden-spike transformation

it will be part of the extension set. Thus, L-Spike events do not occur twice throughout the backchaining algorithm. Their total contribution throughout planning is $O(\ell^3 \log \ell)$.

When an E-Spike-In event occurs, two components of the preimage merge. However, since the "tip" of one component will be inside the other, there is no need to perform this merger. We might as well keep the descriptions of the two components separately. When a spike lies within a component of the preimage, we call it a *hidden spike*. The description of the preimage that we get from the sweep-line algorithm can be changed according to the hidden spike idea with the following transformation. Let preimage arcs be represented by the letter `d` and a subscript that identifies the disk that contains the arc. Let also left (resp. right) edges of the preimage be represented by the letter `l` (resp. `r`) and a subscript that identifies the disk from which the preimage stems. If one of the lists describing the boundary of the preimage has the form $(\alpha, \texttt{d}_i, \texttt{l}_j, \beta, \texttt{r}_k, \texttt{d}_l, \gamma)$, such that:

- α, β, and γ are non-empty sublists,
- the endpoints of $\texttt{l}_j$ and $\texttt{r}_k$ 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 transformation illustrated in Figure 4.13. 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 counterclockwise 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.

In practice, we keep track of the edge endpoints on the boundary of every landmark area. After the line-sweep is completed, we sort these points in counterclockwise cyclic lists (one per landmark area). 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 should form a hidden spike, so we apply the transformation described in the previous paragraph. The combined cost of all transformations is $O(\ell \log \ell)$.

During the angular sweep, we have to be careful to create and delete hidden spikes when necessary. For example, at an E-Include event, if the exiting left ray formed a hidden spike, that spike must be now deleted. Similarly, at an E-Leave event the reaching right ray may form a new hidden spike in the disk it enters. These computations do not affect the complexity of processing each event.

The remaining high complexity contributors (O-Spike events and catastrophic events) constitute a problem that bears a striking similarity with the problem solved by Briggs in [8]. In [8] , O-Spike events are called *vertex critical* and are shown to be uniquely attributable to one of the other $O(\ell^2)$ events; hence, their number is also in $O(\ell^2)$. Furthermore, it is shown that the total number of changes in the preimage because of catastrophic events is not cubic, but quadratic. Using special data structures that store a sorted list of the intersections of each ray with other rays and disks, it is possible to compute the change at each catastrophic event in time linear in the number of changes, without having to call the sweep-line algorithm. We conjecture that similar reductions can be found for our problem too. If this conjecture proves to be true, then the complexity of the algorithm will be reduced by one order of magnitude to $O(\ell^3 \log \ell)$.

4.7 Implementation and examples

We implemented the planning algorithm, along with a robot simulator, in the C language on a DEC-5000 workstation. In the examples that follow obstacle disks are black, landmark disks are white (if they are part of the plan) or grey (if they are not involved in the plan), the initial region is marked with $\mathcal{I}$, and the goal region is marked with $\mathcal{G}$.

Figures 4.14-4.17 present an example with 51 landmark disks, no obstacle disks, a single initial-region disk and a single goal disk. In Figure 4.14 the control uncertainty θ has been set equal to 0.1 radian. The planner returns success after 2 iterations in 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 extension. Although the disk marked E is along the path between D and F, it is not in the termination set of the I-command executed from D. The robot traverses E without shifting to another motion command.

In Figure 4.15 the control uncertainty is 0.15 radian. It takes three backchaining steps of the planner, and 5 seconds of computation, before the initial region can be included in a preimage. In the process, the planner attaches motion commands to the majority of the landmark disks. At execution time, the robot moves first into landmark A, and from there it is able to get into D and eventually to the goal.

In Figure 4.16 we set θ to 0.2 radian. It takes 4 iterations of the planner, and 18 seconds of computation, before the initial region can be included in a preimage. In the process, the planner attaches 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 extension. 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.

Finally, in Figure 4.17 θ is 0.3 radian. A plan is generated after 6 iterations, and 45 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

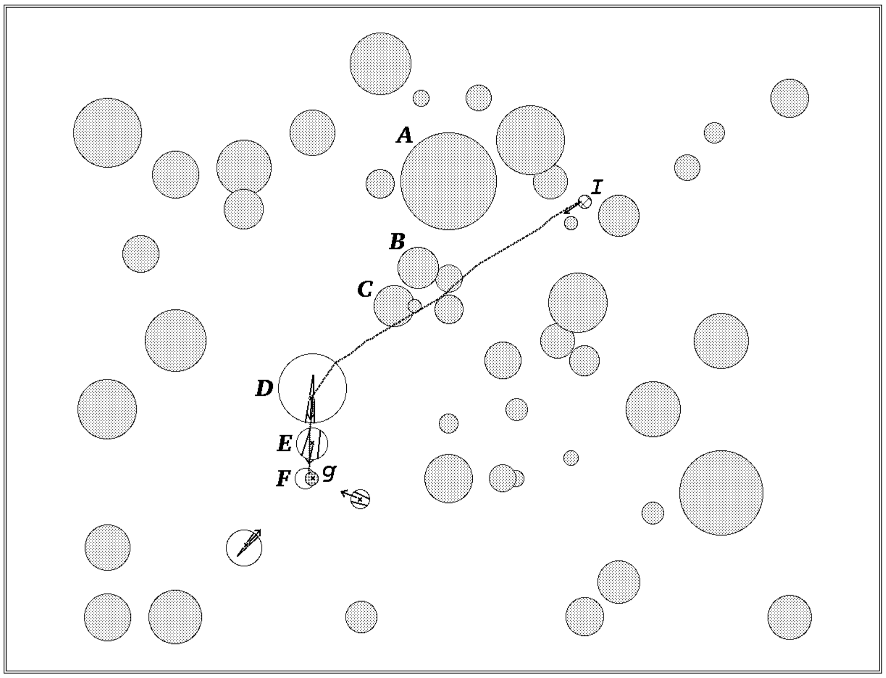


Figure 4.14: Example with $\theta = 0.1$ radian

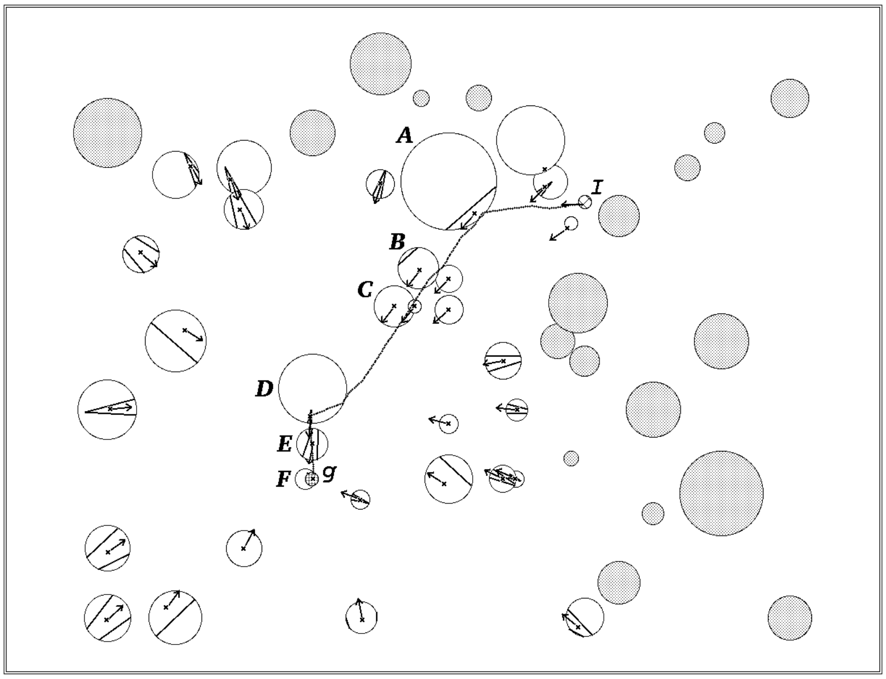


Figure 4.15: Example with $\theta = 0.15$ radian

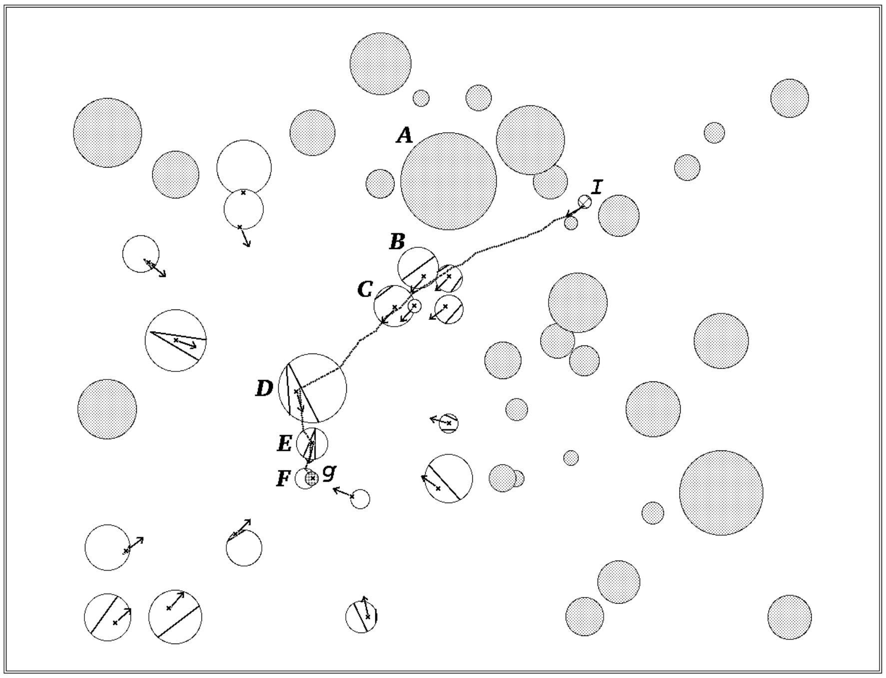


Figure 4.16: Example with $\theta = 0.2$ radian

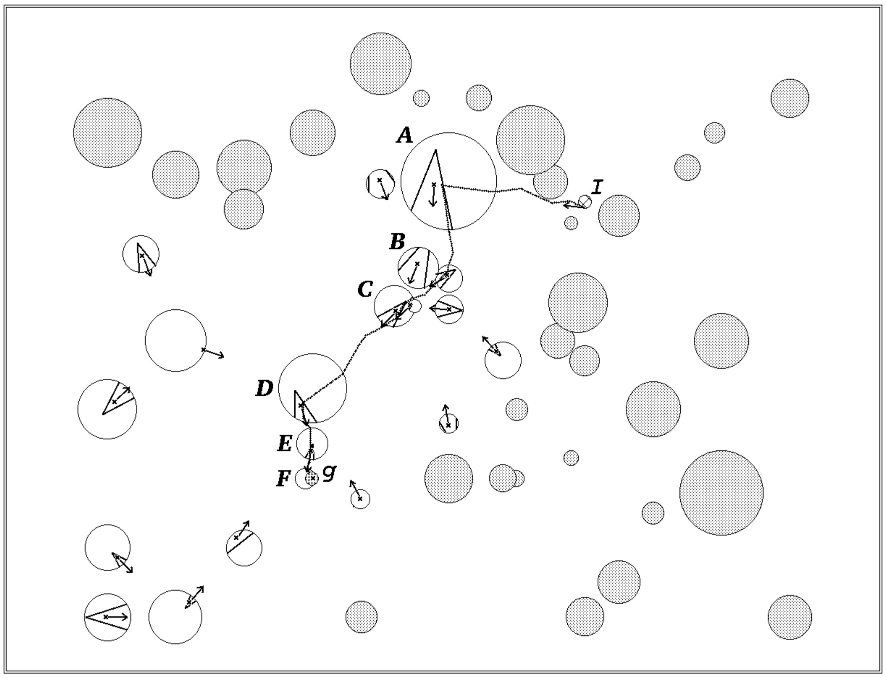


Figure 4.17: Example with $\theta = 0.3$ radian

from the plans found for smaller values of θ. 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.

Another example is shown in Figure 4.18. In this example there are 23 landmark disks forming 19 landmark areas, and 25 obstacle disks forming 13 obstacle areas. The directional uncertainty θ is 0.06 radian. In this case the robot is able to move to disk A, then to K, and from there get to the disk N that intersects the goal. This plan can be found in approximately 30 seconds. In Figure 4.19 we present the same example for θ equal to 0.1 radian. The robot is no longer able to pass reliably through the obstacles, so it is forced to take the long outside route. This plan is discovered after 13 iterations of the planner (it takes about 4 minutes), and the shown execution trace has also 13 steps.

Figure 4.20 shows a corridor of 15 obstacle and 15 landmark disks. The robot stands in front of this corridor and wishes to get to the goal that lies at the other end of the corridor inside a big landmark region. The robot starts out with a zig-zag motion among the obstacles, but as soon as it discovers that it is safe to move straight to the big landmark it does so. This example demonstrates the property of our planner to produce plans with the minimum possible number of steps. The plan is found after 5 iterations in less than 30 seconds.

Figure 4.21 shows a maze of 77 obstacles with 5 landmark disks scattered in the maze, and two landmark disks outside the maze. The robot wishes to traverse the maze, in order to get into the goal on the other side. Planning takes about 3 minutes and a 7-step plan is constructed allowing the robot to move safely into the goal.

4.8 Properties of the motion plans

The planner we described in the previous sections has several desirable properties:

Soundness By definition, the planner creates only correct plans (i.e., plans guaranteed to succeed if the assumptions made in the problem statement hold). Therefore

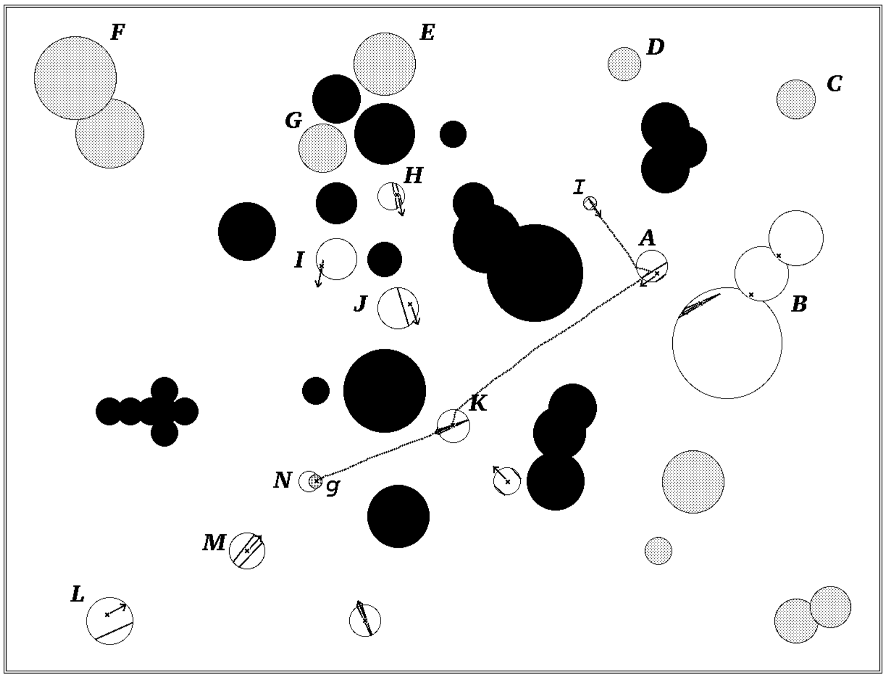


Figure 4.18: Example with obstacles with $\theta = 0.06$ radian

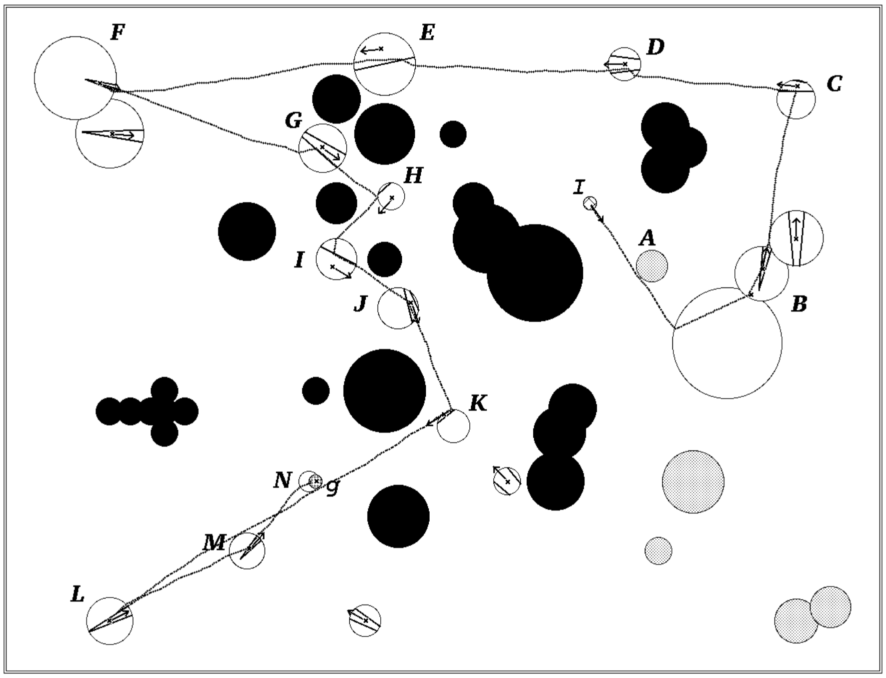


Figure 4.19: Example with obstacles with $\theta = 0.1$ radian

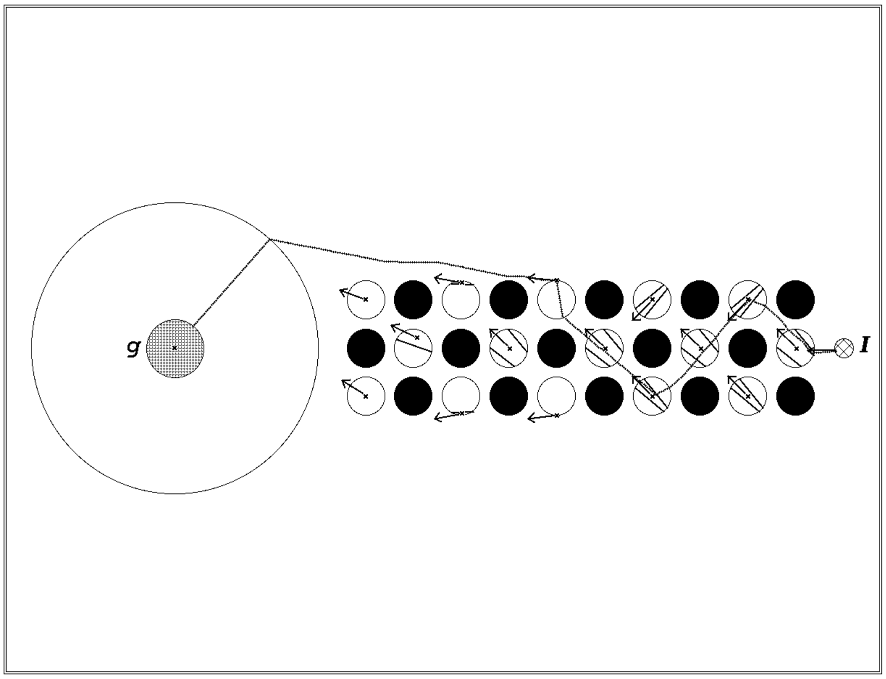


Figure 4.20: Traversing a corridor of obstacles

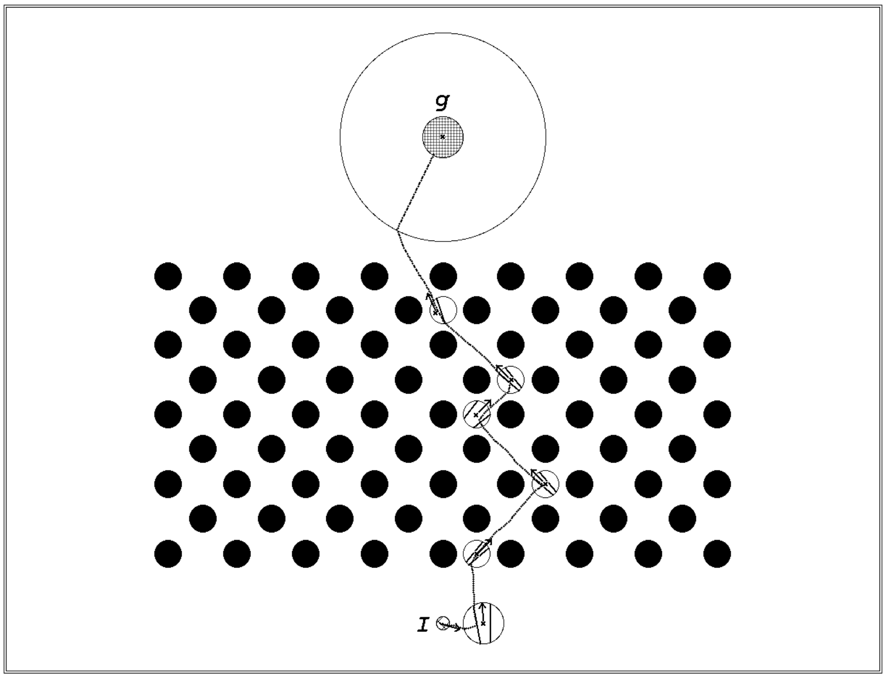


Figure 4.21: Traversing a maze of obstacles

the planner is sound.

Completeness At each backchaining step we identify all configuration space points that can achieve the current extension set in one step (the combination of a P-command and an I-command). Therefore, during the i-th backchaining step the extension set holds the maximal set of points that can get to the extension of the goal in at most $i-1$ steps. If a plan with n steps exists it will be found by the planner after n iterations. If no plan exists, the planner will return failure after a number of iterations that is bounded by the number of landmark areas. Thus, the planner is complete.

Optimality We consider a motion plan to 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 $\mathcal{G}$. The maximal number of steps for a plan produced by our planning algorithm is equal to the number of backchaining iterations before an extension set or a preimage contains the initial region. By definition of the omnidirectional preimage, 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. In addition, 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.

Polynomiality The polynomial time bound of the algorithm permits fast execution even in complicated environments (like the examples presented in the previous section).

Distribution over landmark disks A major characteristic of the plans created by the planner is their distributed nature. Commands, stored locally at the landmark disks, provide a way of reliable navigation from landmark area to landmark area, and are almost independent from the initial region or the destination of the robot. Indeed, we can create one plan to reliably achieve each landmark area from all points of the workspace from where this is possible. Thus, we cover all possible planning

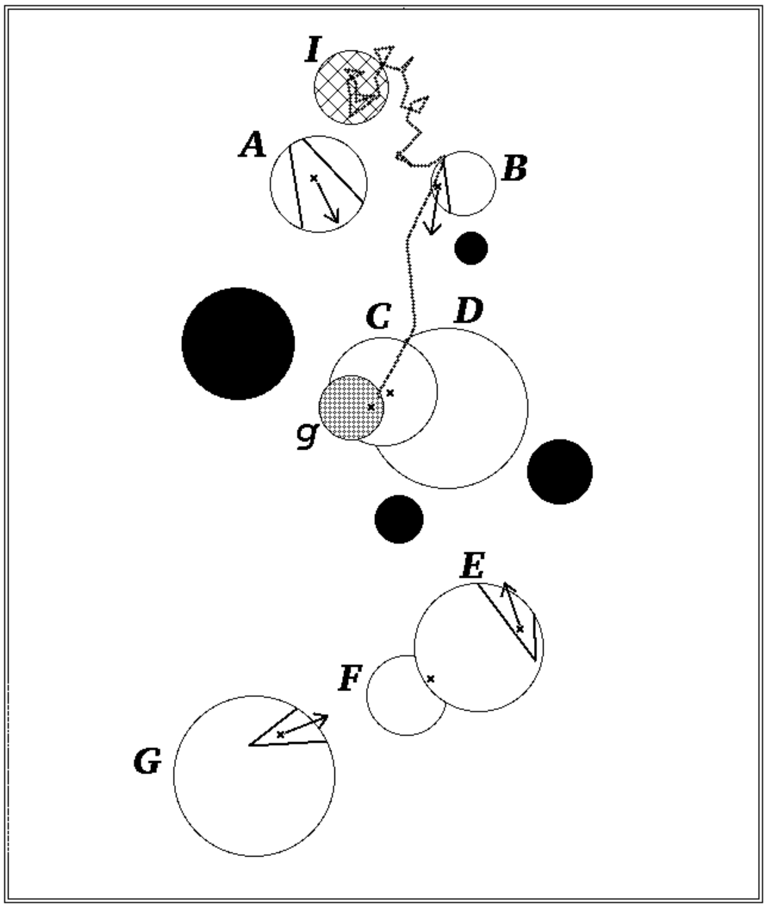


Figure 4.22: Initial random motion

situations that may arise in this workspace with $O(\ell)$ local commands per landmark disk. The total size of the stored information is quadratic instead of exponential (which is usually the case in such universal problems). For a detailed analysis of this technique see Section 5.4.

Robustness with respect to failures Up to now we have only considered examples where the planning algorithm returns success. However, even when no guaranteed plan exists, our planner computes an incomplete plan that may be useful. This plan does not associate an I-command with the initial-region, but it creates motion commands for all the landmark disks in the final (largest) extension considered. If, somehow, the robot gets into one of these disks, then it has a guaranteed plan to get into the goal. This incomplete plan is "maximal" in the sense that it attaches a guaranteed plan to achieve the goal to every single point in the configuration space, for which such a guaranteed plan exists.

One way of letting the robot try to reach one of these disks is random (Brownian) motion with reflection on the boundaries of the workspace and the obstacles. Consider again the example of Subsection 4.3.1, and let the directional uncertainty have a higher value so that no guaranteed plan exists. In Figure 4.22 we see that a motion command has been associated with every landmark disk in the workspace, but it has not been possible to construct an I-command that is guaranteed to bring the robot into one of the landmark disks. Initially, the robot is instructed to perform a random motion according to a two-dimensional random walk. It moves aimlessly for a while until it hits disk B; from there it gets to the goal using the computed plan. The Brownian motion in a bounded subspace of the plane, is guaranteed at some point to hit one of the white disks. However, the length of the random motion is unbounded. The bigger the area of the white disks with respect to the rest of the workspace, the smaller the expected duration of the random motion. By construction, our planner computes the maximal such region.

The same idea may be used to help the robot deal with unexpected failures (e.g. a landmark is turned off, the robot is pushed outside a landmark area, etc.). Again a random motion is guaranteed to bring the robot into a landmark area (possibly different from the one it was looking for) with a guaranteed plan to the goal.

In Chapter 6 we present a more intelligent way to select an initial non-guaranteed command. We show that this command maximizes some measure of the likelihood that the robot will enter one of the landmark disks that have an associated guaranteed plan (i.e., one of the white disks).

Chapter 5

The Limits of Polynomiality

Despite the nice properties of the algorithm presented in the previous chapter, there is still a major question to be answered: Is the algorithm applicable to real-world situations, and, if yes, what is the cost of engineering the robot and its workspace, so that the assumptions of the algorithm are satisfied? If an assumption is too restrictive (or too costly to implement), we would like to eliminate it, without sacrificing the soundness, completeness, or polynomiality of the planning algorithm. In this chapter we investigate the effect of the elimination of several of the assumptions of the previous chapter. In some cases, the elimination has no significant effect; in others, it makes the problem intractable. Throughout this chapter, d denotes the commanded direction of motion, and θ the directional uncertainty.

5.1 Landmark and obstacle geometry

So far, we have assumed that the shape of all configuration space objects (initial, obstacle and landmark regions) is circular. Even though we can approximate any other shape with overlapping circular disks, we often need a big number of such disks in order to get a reasonably good approximation. Fortunately, it is not hard to extend the algorithm to the case where the objects are generalized polygons, i.e., regions of the workspace bounded by straight edges and circular arcs. Such a configuration space corresponds to a workspace model of a circular robot navigating in a polygonal

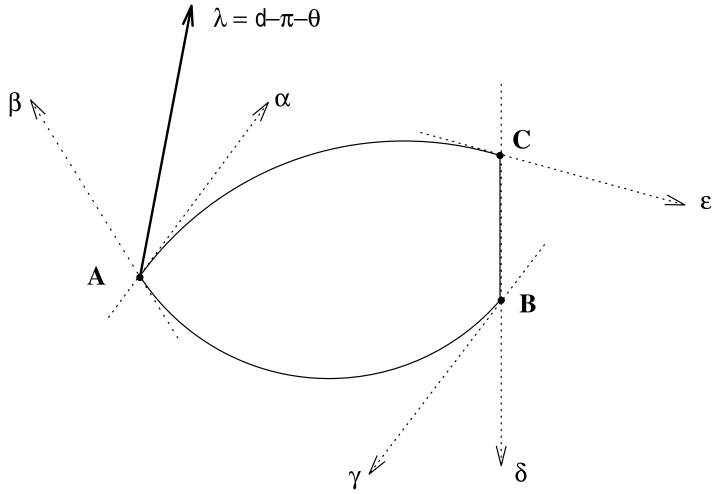


Figure 5.1: An anchored ray

world. This model is usually sufficient for most practical applications.

In this case, the computation of the directional preimage remains unchanged. The straight edge segments are actually easier to handle than circular arcs. However, we need to add a few additional critical events for the computation of the omnidirectional preimage. In the circular case, rotating rays slide smoothly on the perimeter of landmark and obstacle disks remaining at all times tangent to them. A ray sliding on a curve, which contains points where the curve is not differentiable, (e.g., a generalized polygonal curve) will stick at such points for a while, until it is ready to slide onto the next edge of the curve. As an example, consider Figure 5.1. A left ray with orientation $\lambda = d - \pi - \theta$ is shown anchored at vertex **A**. As d rotates counterclockwise, so does the ray. The ray remains anchored at **A** as long as λ is between α and β. Then it starts sliding on the arc **AB** until λ becomes equal to γ, at which point it sticks to **B**. It remains so until $\lambda = \delta$. At this point the ray is parallel to the straight edge **CB**. A further counterclockwise motion of the ray causes its origin to jump to vertex **C**, where it remains until $\lambda = \epsilon$. Finally, the ray slides on the arc **CA** until it sticks once again on **A** for $\lambda = \alpha$.

A ray which rotates about a fixed origin is called an *anchored ray*. An *E-Left-Anchor* (*E-Right-Anchor*) critical event causes a sliding left (right) ray to become anchored. An *E-Left-Release* (*E-Right-Release*) critical event causes an anchored left

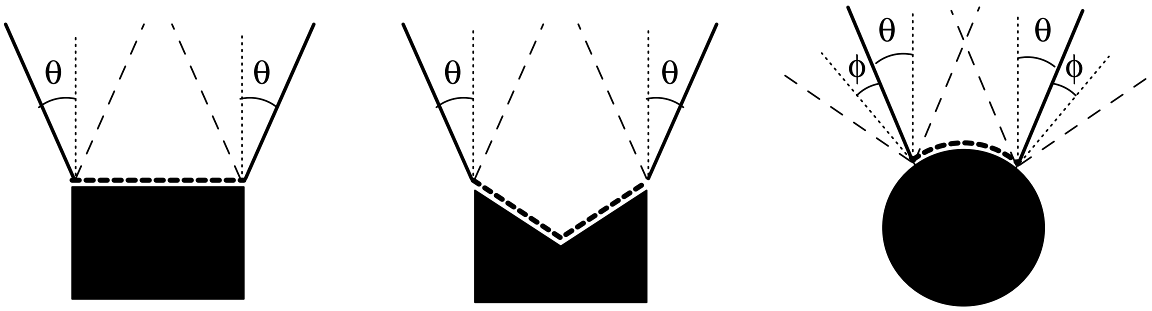


Figure 5.2: Rays stemming from sticky points

(right) ray to start sliding on an arc. At straight edges, an E-Release critical event at one vertex coincides with an E-Anchor event at the other vertex. In this case, the two events are combined into a single event. Such an event is called *E-Left-Align* (*E-Right-Align*), since it occurs when a ray becomes aligned with an edge of a landmark or obstacle region. In the example of Figure 5.1, we have E-Left-Anchor events for $d = \pi + \alpha + \theta$ and $d = \pi + \gamma + \theta$, E-Left-Release events for $d = \pi + \beta + \theta$ and $d = \pi + \epsilon + \theta$, an E-Left-Align event for $d = \pi + \delta + \theta$, and similar events for a right ray.

Anchored rays can be thought of as tangent to disks of zero radius, so all the calculations for sliding rays work also for anchored rays. As a matter of fact, most of the times it is easier to work with anchored rays; most notably, the locus of the spike of two anchored rays is not the fourth degree curve described in Appendix A; it is simply a circular arc.

If ℓ is the number of features of the configuration space (edges and vertices), the total number of E-Anchor, E-Release and E-Align events is $O(\ell)$. Each such event can be processed in constant time, so the complexity of the algorithm with respect to the complexity of the configuration space is not affected.

5.2 Compliant motions

Another assumption that is not too difficult to remove, is the one that forbids the robot to slide on obstacle surfaces. Although sometimes sliding is not desirable, often it enhances the navigational efficiency of a robot (see Chapter 3). Allowing compliant motions enlarges the preimage of a given goal.

Let us assume that the coefficient of friction is uniform and equal to $\tan\phi$, where ϕ is the half angle of the friction cone on all obstacle surfaces. In this case, rays stemming from obstacles are not tangent to the obstacles, but their origins are the endpoints of the *sticky part* of the obstacle boundary (*sticky points*). The sticky part, is the part of an obstacle where there is a possibility that the robot may stick due to obstacle geometry or friction. Some examples are shown in Figure 5.2, where the sticky part is drawn with a thick dashed line. In (a), the upper edge of a rectangular obstacle is sticky and must be avoided. In (b), the robot may stick in the triangular depression of a concave obstacle. In (c), the sticky part of a circular obstacle lies between the points where the friction cone and the inverted velocity cone are tangent to each other without overlapping.

Rays stemming from sticky points behave like regular rays. They slide on circular arcs[1], they get anchored at obstacle vertices, and they jump suddenly from a vertex of a straight obstacle edge to the other vertex of the edge. These sudden jumps occur when an edge becomes part of the sticky part of the obstacle (in which case we have an *E-Stick* event), or when an edge stops being sticky (an *E-Slide* event). The E-Stick and E-Slide events are very similar to the E-Align events of the previous section. In fact, when there is no friction, the E-Stick event degenerates to an E-Left-Align event, and the E-Slide becomes an E-Right-Align event. The number of E-Stick and E-Slide events is linear with the complexity of the workspace, and each such event can be processed in constant time.

5.3 Allowing contact between obstacle and landmark disks

Ideas similar to the ones presented in the previous subsection can be used to remove the assumption that obstacle and landmark regions are not in contact with each other. This assumption is often too restrictive, especially in the case where workspace

---

[1] The angular position of the sticky points of a circular obstacle is given by $d + \pi \pm (\theta + \phi)$

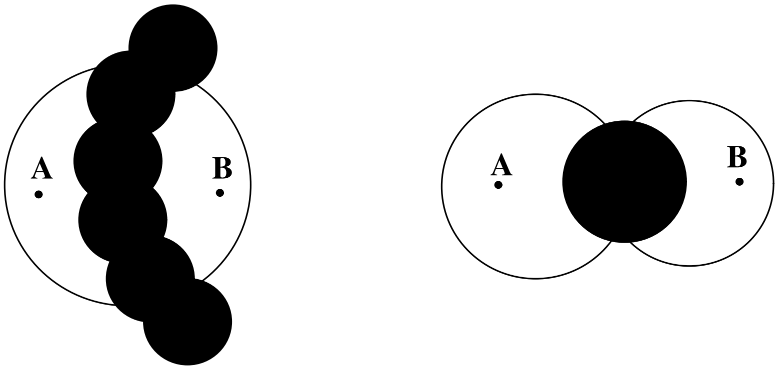


Figure 5.3: Problem arising from overlapping landmark and obstacle regions

obstacles play the role of landmarks. A landmark region and an obstacle region that are caused by the same physical object usually overlap.

If we allow overlapping obstacle and landmark regions we introduce two kinds of complications. First, navigation between any two points in a landmark region, or intersecting landmark regions may no longer be possible (see Figure 5.3). This can be addressed by requiring that landmark regions do not overlap with obstacles, although they can be in contact with them.

The second complication has to do with the computation of the preimage. The vertices of the contact edges between landmark and obstacle regions (*contact vertices*) play an important role in this computation. If sliding on the obstacles is not allowed, then a ray must be erected at contact vertices, in order to make sure that the robot will not hit the obstacle. Rays erected at contact vertices are anchored.

The example of Figure 5.4 shows the behavior of left and right rays when their origin hits a contact vertex. In 5.4(a), the commanded direction d is such that the preimage is not affected by the existence of the obstacle. When the origin of the left ray hits the upper contact vertex (i.e., the slope of the left ray becomes α, so $d = \pi + \alpha + \theta$), we have an E-Left-Anchor event. The left ray rotates anchored at its origin (5.4(b)), until it becomes parallel to β, at which point we have an E-Left-Release event. The ray now starts sliding on the obstacle surface (5.4(c)), until it is reduced to zero length pushed by the right ray on the obstacle surface. This is the familiar E-Spike-Death event, which occurs when the right ray is aligned with

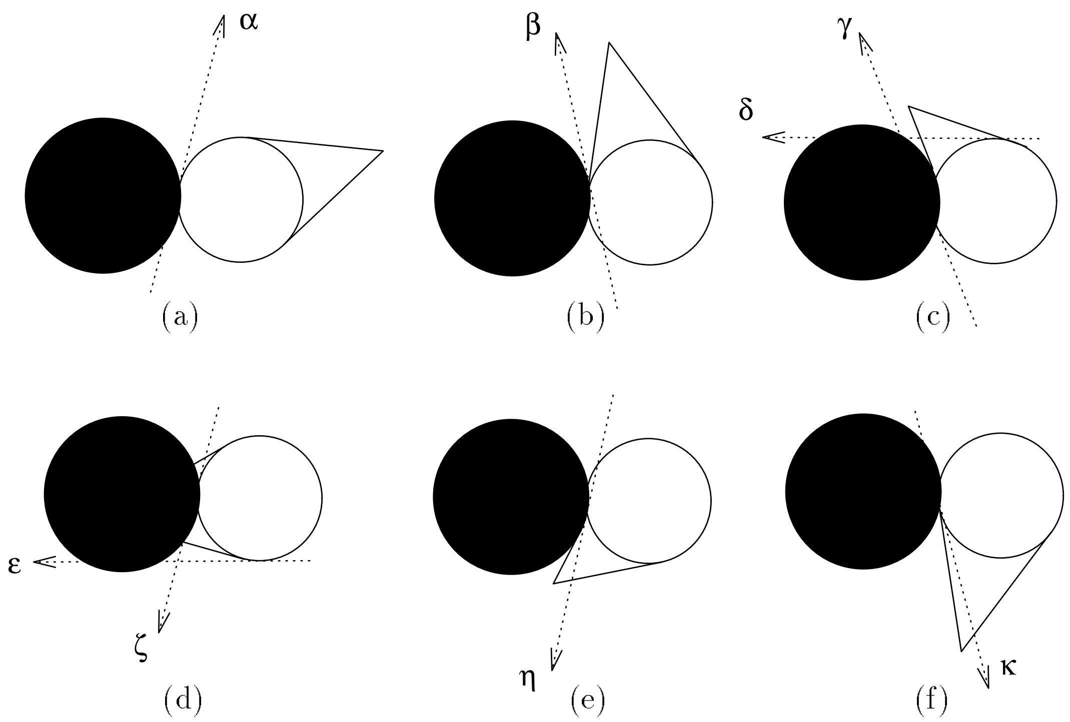


Figure 5.4: Critical events at contact vertices

δ (i.e., $d = \pi + \delta - \theta$). In the mean time, when the direction of left rays becomes equal to γ, a left ray emerges at the other contact vertex during an E-Left-Birth. In Figures 5.4(d)-(f) the symmetric events occur. The right ray vanishes at the upper contact vertex (E-Right-Death event at $d = \pi + \zeta - \theta$), a spike emerges (E-Spike-Birth event at $d = \pi + \epsilon + \theta$), the right ray of the spike gets anchored at the lower contact vertex (E-Right-Anchor event at $d = \pi + \eta - \theta$), and, finally, the anchored right ray starts sliding on the landmark disk surface (E-Right-Release event at $d = \pi + \kappa - \theta$). It is straightforward to compute the directions α, β, γ, δ, ϵ, ζ, η and κ, and from them the critical orientations of the commanded velocity.

Figure 5.5 shows a slightly different example. The left ray of the spike gets anchored at the upper contact vertex, as before, when its slope becomes α (5.5(a)). However, in this case the right ray pushes the left ray to zero length before the left ray gets a chance to start sliding on the obstacle surface (i.e., $\gamma - \theta$ precedes $\beta + \theta$, so $\gamma - \beta < 2\theta$). Thus, we get an E-Spike-Death event while the left ray of the

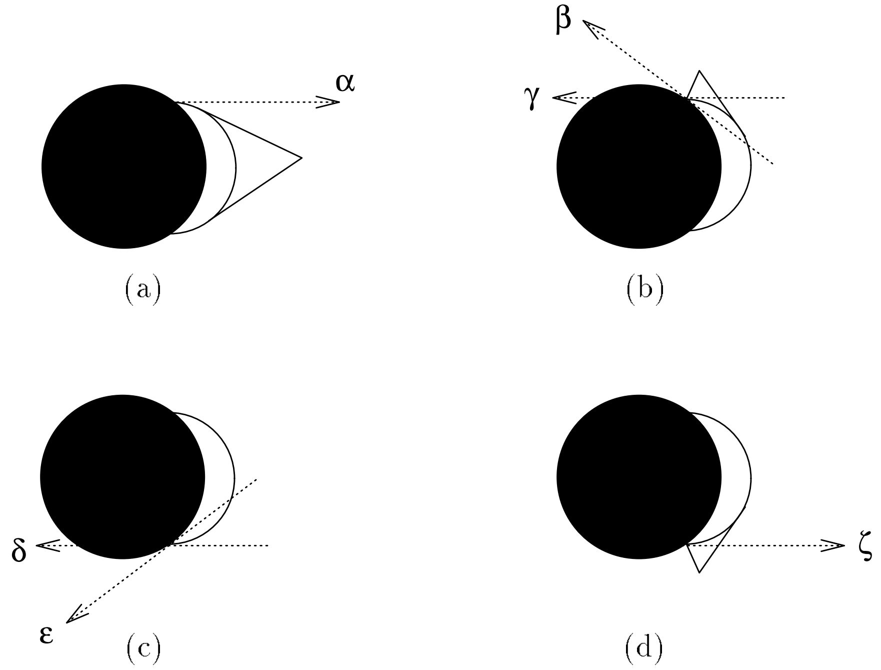


Figure 5.5: Birth and death of a spike with an anchored ray

spike is still anchored (Figure 5.5(b)). An E-Spike-Birth event occurs when a spike emerges with its right ray anchored at the lower contact vertex for $d = \pi + \delta + \theta$ (Figure 5.5(c)). Finally, the anchored right ray is released during an E-Right-Release event at $d = \pi + \zeta - \theta$ (Figure 5.5(d)).

Similar calculations allow us to deal with generalized polygonal obstacles and landmarks in contact, as well as compliant motions on obstacle surfaces. As a result, the algorithm given in Chapter 4 can be extended to deal with generalized polygonal shapes, compliant motions and contact between obstacle and landmark regions, without losing soundness, completeness, or polynomiality.

5.4 Universal plans

When there is need for frequent planning in a workspace with fixed landmark and obstacle regions, it is possible to save planning time, by precomputing and storing a guaranteed plan between any two landmark areas for which such a guaranteed plan exists. Given a planning problem specified by an initial region $\mathcal{I}$ and a goal region $\mathcal{G}$, we only need to lead the robot from $\mathcal{I}$ to a landmark area, from where there exists a known guaranteed plan to another landmark area that intersects $\mathcal{G}$.

5.4.1 Precomputation

Let us index each landmark area LA in the workspace with a distinct integer $i \in [1, s]$. Then, for each LA_i we compute a *maximal plan* $\mathbf{U}_i$ by invoking the planner with LA_i as the goal and a dummy initial region that cannot be contained in any preimage. The algorithm backchains until the extension set cannot grow any more. (The notion of maximal plans was introduced in 4.8.) All maximal plans are combined in a data structure, the *universal plan* $\mathbf{U} = (\mathbf{U}_1, \mathbf{U}_2, \ldots, \mathbf{U}_s)$.

The function $disks(\mathbf{U}_i)$ returns the set of landmark disks, from where there exists a guaranteed plan to landmark area LA_i. All these plans have already been found, and they constitute the maximal plan $\mathbf{U}_i$. Each landmark disk may belong to several sets $disks(\mathbf{U}_i)$ (as many as the distinct landmark areas which are achievable from this disk with a guaranteed plan). Therefore, at each landmark disk several motion commands are stored. Each such motion command is marked with the index i of the landmark area that is the goal of the maximal plan $\mathbf{U}_i$ containing this motion command. In addition, each motion command in $\mathbf{U}_i$ is annotated with the maximum number of steps needed to achieve the goal (using $\mathbf{U}_i$) from the landmark disk where the command is attached.[2]

5.4.2 Planning

Given a planning problem specified by an initial region $\mathcal{I}$ and a goal region $\mathcal{G}$ in a preprocessed workspace, we proceed as follows. First, we identify the landmark areas which intersect the goal, and create a set $\mathbf{M}(\mathcal{G})$ containing all landmark disks from which there exist guaranteed plans to one of these landmark areas. If the initial region lies entirely within landmark disks belonging to $\mathbf{M}(\mathcal{G})$, then one (or more)

---

[2] This number is equal to the number of backchaining steps it took to reach the disk when creating $\mathbf{U}_i$.

plans are readily available without further computation. Otherwise, we compute the omnidirectional preimage of all disks in $\mathbf{M}(\mathcal{G})$. If the initial region can be contained in this preimage, then a plan has been found; otherwise the planner returns failure. No backchaining is needed.

In order to prove that no backchaining is needed, consider the case where $\mathcal{G}$ intersects two landmark areas LA_a and LA_b. Let us assume that there exists a direction d^* for which the preimage of $disks(\mathbf{U}_a) \cup disks(\mathbf{U}_b)$ intersects a landmark disk L which does not belong either to $disks(\mathbf{U}_a)$ or to $disks(\mathbf{U}_b)$. For this to happen, the preimage component that intersects L must contain at least one disk that belongs to $disks(\mathbf{U}_a) \setminus disks(\mathbf{U}_b)$ and at least one disk that belongs to $disks(\mathbf{U}_b) \setminus disks(\mathbf{U}_a)$. Otherwise, the component would consist solely of disks belonging to either $disks(\mathbf{U}_a)$ or $disks(\mathbf{U}_b)$. This is not possible, because the preimage of $disks(\mathbf{U}_a)$ (as well as the preimage of $disks(\mathbf{U}_b)$) cannot intersect any other landmark disks (otherwise backchaining would not have terminated during the precomputation of $\mathbf{U}_a$ or $\mathbf{U}_b$).

We have thus established that the preimage component that intersects L must contain disks that do not belong to $disks(\mathbf{U}_a)$ and disks that do not belong to $disks(\mathbf{U}_b)$. But this is not possible either. Were it true, it would be the case that either the preimage of $disks(\mathbf{U}_a)$ would intersect a disk not in $disks(\mathbf{U}_a)$, or the preimage of $disks(\mathbf{U}_b)$ would intersect a disk not in $disks(\mathbf{U}_b)$. In either case, backchaining (during the precomputation of $\mathbf{U}_a$ or $\mathbf{U}_b$) would not have stopped, a contradiction. Thus, no new landmark disks can be intersected by the preimage of $disks(\mathbf{U}_a) \cup disks(\mathbf{U}_b)$, and, therefore, backchaining is not needed.

5.4.3 Plan Execution

If the robot is not already in one of the landmark disks in $\mathbf{M}(\mathcal{G})$, it executes the motion command attached to the initial region. When it finds itself in a landmark disk of $\mathbf{M}(\mathcal{G})$, the robot must select one of the commands that are attached to this disk. The danger with selecting commands from different plans is the possibility of entering an infinite loop of motion commands.

Consider the example of Figure 5.6. There are four landmark disks A, B, C and D, each constituting a separate landmark area. The goal intersects only A and D.

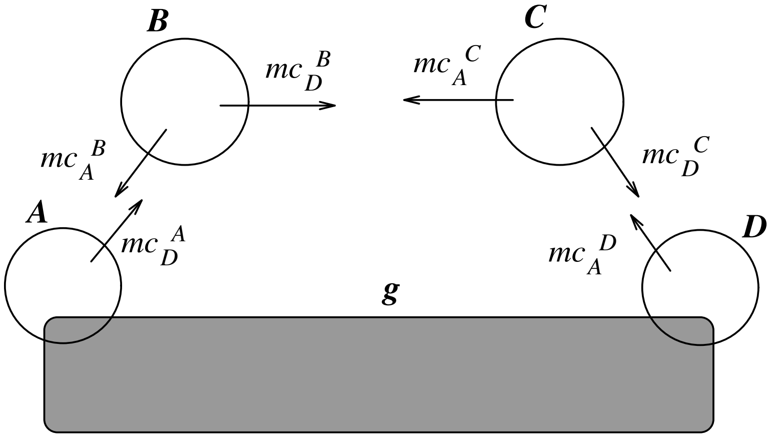


Figure 5.6: A possibly cyclic plan execution

In the figure we show only the commands that belong to plans $\mathbf{U}_A$ and $\mathbf{U}_D$. Assume that the robot starts from landmark disk B. It has the choice of two commands, mc_A^B which leads directly to A and mc_D^B which leads to disk C. If for some reason it selects the latter and moves to disk C, it has again to choose between two commands, mc_A^C and mc_D^C. If it selects mc_A^C, switching from executing $\mathbf{U}_D$ to executing $\mathbf{U}_A$, it goes back to disk B. If the criteria for choosing commands are deterministic and do not change over time, then the robot will be stuck in the above loop forever.

The easiest way to avoid cycles during plan execution is to stick to a particular plan (select motion commands with the same subscript), but then plan execution may become unnecessarily long. For example, look at Figure 5.7. The goal intersects landmark disks A and E, so we show only plans $\mathbf{U}_A$ and $\mathbf{U}_E$. For each motion command we also show the maximum number of steps that it may take the robot to achieve the goal of the plan, where the command belongs. The preimage of A, B and C intersects disk D, so there exists a guaranteed plan from D to A, whose execution may take at most three steps. The preimage of E and C does not intersect D. A robot that starts from D has to follow the single command mc_A^D. Given the policy to stick to motion commands with the same subscript, if the robot gets to disk C, it will follow command mc_A^C to get to B, and from there to A and to the goal. The number of steps in this execution trace is three. If instead at disk C the robot decides to go with command mc_E^C, it will get to disk E and to the goal immediately. The number

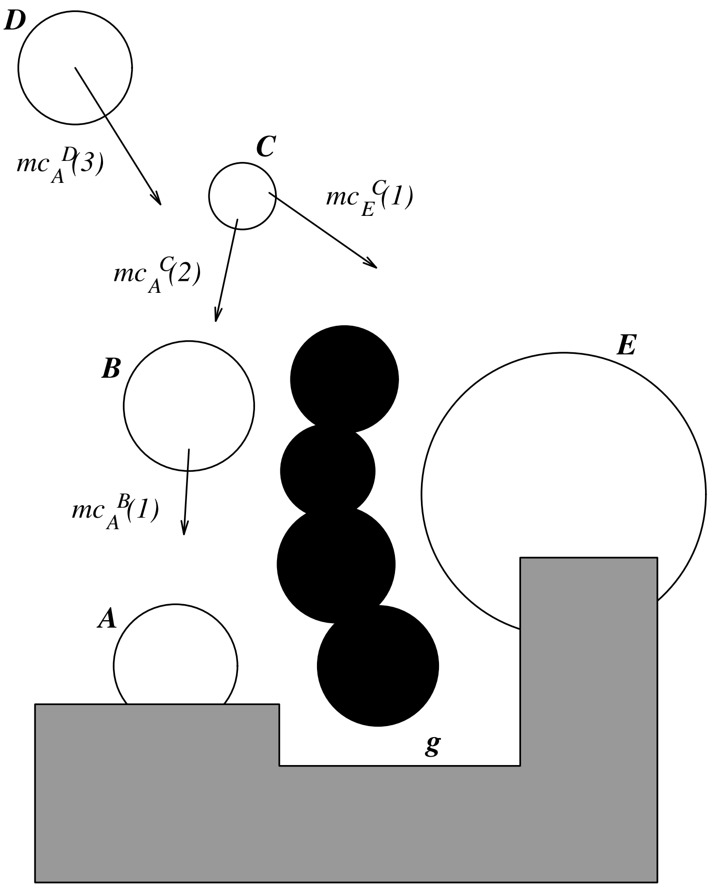


Figure 5.7: Using the minimum number of execution steps

of steps of this execution trace is two.

It is possible to minimize the number of execution steps, and at the same time avoid cycles altogether. At each landmark disk, the robot selects the motion command that promises the least maximum number of steps to the goal.[3] The "distance" of the robot to the goal measured in steps strictly decreases with each execution step, therefore the robot may not enter a cycle. In the above example, command mc_E^C (promising at most one step) is preferable to mc_A^C which may take up to two steps. Thus the robot will move from D to C, then to E and into the goal.

---

[3] The maximum number of steps to the goal for each motion command is equal to the number of backchaining steps it took to find the particular command while computing the complete plan.

5.4.4 Complexity

If l is the number of landmark and obstacle disks and s is the number of landmark areas, precomputation takes $O(s\ell^3 \log \ell)$ per landmark area, for a total of $O(s^2\ell^3 \log \ell)$. Planning takes $O(\ell \log \ell)$ to compute the intersection of $\mathcal{G}$ with the landmark disks and create $\mathbf{M}(\mathcal{G})$, $O(\ell \log \ell)$ to check whether $\mathcal{I}$ is completely included in the disks of $\mathbf{M}(\mathcal{G})$, and, if it is not, another $O(\ell^3 \log \ell)$ to find the omnidirectional backprojection of the disks in $\mathbf{M}(\mathcal{G})$.

5.5 Uncertainty in landmark areas

Perhaps the most disturbing assumptions in the previous chapter are those used in the definition of the landmark areas, namely, that control and position sensing errors are null within these areas, while position sensing is inexistent outside.

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 (i.e., "landmarks") are visible by the sensors. Then the robot knows its position with good accuracy. When no or few features are visible, the robot mostly 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 usually 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. (The navigation system does not have to use the same model as the planner; it may use a more sophisticated one, if this is possible.) In the worst case, the no-sensing assumption outside landmark areas may only lead our planners to return failure, while reliable plans exist in practice and, possibly, could have been found by more powerful planners able to deal with more sophisticated models.

The assumption that control is perfect in the landmark areas is rather liberal. We believe, however, that it is a reasonable one, provided that we choose safe features and equip the robot with the right sensors. Landmark areas with sharp boundaries can be obtained by introducing artificial landmarks (e.g., radio or magnetic beacons) and/or

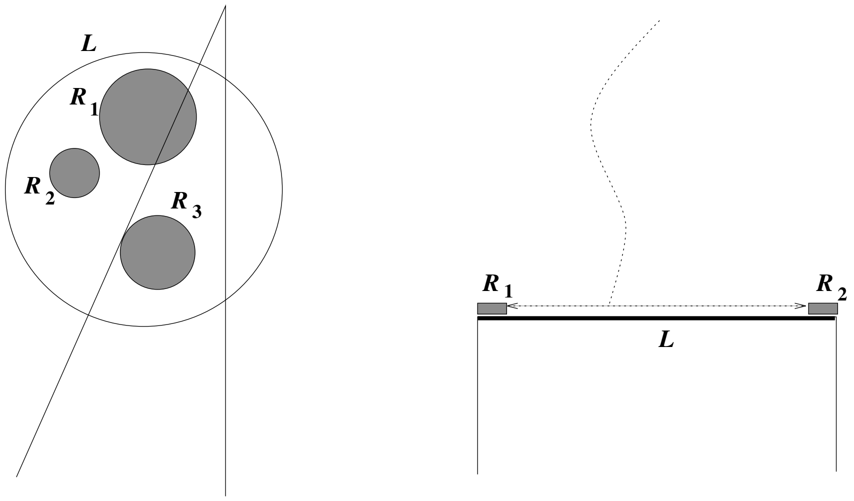


Figure 5.8: Generalized landmarks

thresholding an estimate of the sensing uncertainty. For example, the notion of a "sensory uncertainty field" (SUF) is introduced in [91]. At every possible point q in the configuration space, the SUF estimates the range of possible errors in the sensed configuration that the navigation system would compute by matching the sensory data against a prior model of the workspace, if the robot were at q. The SUF is computed at planning time from a model of the robot's sensing system. Thresholding it yields landmark areas. Uncertainty in the location of a landmark and/or fuzziness of its boundary can be handled by defining a smaller landmark area for our planner.

It should also be noted 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 imperfect-control command. This region is the intersection of the backprojection that yielded the command with the landmark area. Position sensing uncertainty in a landmark area could be half the radius of the largest disk fully contained in the exit region of the landmark area without putting plan execution at risk. Thus, although the planner assumes perfect sensing in landmark areas, we can create these areas by engineering the workspace so that the sensors provide just the information that is needed by the plan (see [36] for a similar idea).

5.5.1 Generalized landmarks

The definition of a landmark can be further modified without having to significantly change the planning techniques developed above. For example, we could accept a landmark disk (or generalized polygon) L such that the robot correctly knows at any time if it is inside or outside L, but, if it is in L, it does not know where. If during planning, a backprojection intersects L, this is not sufficient to include L in the extension of the goal. L must be completely contained in the backprojection. The critical events for which this has to be tested are exactly those used to check the containment of an initial-region disk.

The above variant of a landmark can be generalized into the notion of a *generalized landmark*, as follows: Consider a region L, such that if the robot is in L, it knows that it is in L and has a way to accurately reach a subregion R of L. However, the robot may not have perfect control in L, nor perfect position sensing. For example, the landmark may be a wall. When the robot makes contact with the surface of the wall, its bumpers detect contact, but it still does not know precisely its position. By tracking the wall in some given direction, the robot can detect the end of the wall, a point where it knows its position with accuracy. This point (or a small disk around this point) can be considered as a landmark area that the planner will try to enclose in a backprojection. If this happens, the whole region L will be added to the goal extension. Figure 5.8 presents two examples of generalized landmarks. The number of subregions in each landmark can be greater than one, but it has to be bounded in order not to affect the complexity of the planner.

Finally, it is possible to define landmarks hierarchically, so that navigation within landmark regions is based on smaller local landmarks. Thus, our landmark-based navigation algorithm can be called recursively a number of times. In this way, we can consider only a few landmarks each time the algorithm is called for considerable savings of planning time. Hierarchical planning is a very powerful technique used extensively by humans. Real-world robots will have to use it, too.

5.6 Confusable landmarks

Up to now we have assumed that landmarks are unambiguously distinguishable by the robot. Sometimes, though, this is not the case. For example, if wall corners are used as landmarks, most of them look the same and cannot be distinguished from each other. In this case, we say that the workspace contains *confusable landmarks*. The robot may have perfect position sensing relative to a landmark, but if there is uncertainty about the identity of the landmark, no unambiguous global positional information can be deduced.

Confusable landmarks may belong to the same extension set, if we can derive a common "exit strategy" for all of them, i.e., we attach the same (relative) exit region, commanded velocity direction and termination condition to each landmark. If there are intermediate goal landmarks that are confusable with landmarks in the extension set, then these intermediate goal landmark regions should be treated as forbidden regions (the same as obstacle regions), because if the robot gets into them it will terminate its motion prematurely. Whether a larger extension set (along with the corresponding forbidden regions) or a smaller extension set (with fewer or no forbidden regions) is preferable depends on the particular planning problem, something that makes a complete planner exponential.

5.6.1 Finding a common exit strategy

Sensory readings in each landmark region give positional information relative to a frame of reference attached to this region. When global information is not available, perfect-control motion is relative to this local frame of reference.

Consider the case where the goal $\mathcal{G}$ intersects two confusable landmarks L_1 and L_2 (Figure 5.9). Let $R_1 = \mathcal{G} \cap L_1$ and $R_2 = \mathcal{G} \cap L_2$ be the intersection regions of the goal with the landmarks, expressed in the local frames of reference of landmarks L_1 and L_2 respectively. We map the two landmark regions so that their local frames of reference coincide, and we compute the intersection $R_1 \cap R_2$. If this intersection is non-empty, then there exists a common perfect-control motion command that is guaranteed to lead the robot into the goal. This command, expressed in the local

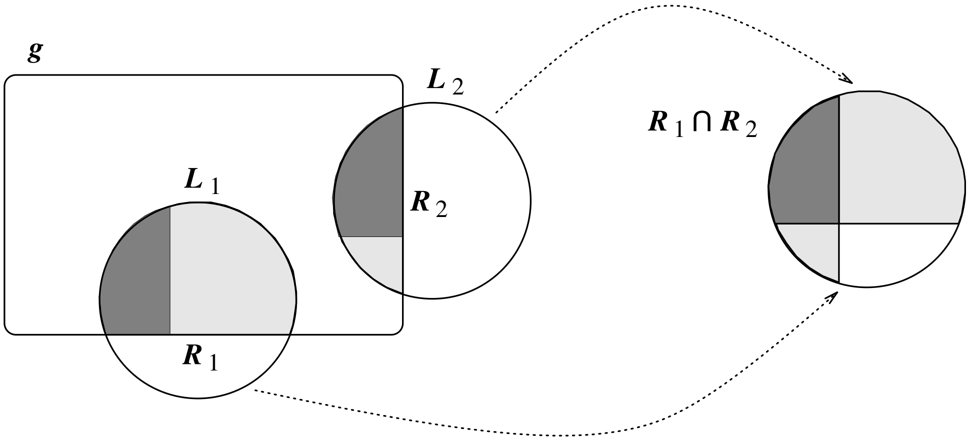


Figure 5.9: Common exit region

coordinate systems, is $(R_1 \cap R_2)$. In the example of Figure 5.9 $(R_1 \cap R_2)$ leads the robot either to the left (darker) part of R_1 or to the upper (darker) part of R_2, in either case into the goal. L_1 and L_2 form the extension of the goal.

If $\mathcal{T}_1$ ($\mathcal{T}_2$) is the transformation from the frame of reference of L_1 (L_2) to the workspace coordinate system, then the possible global coordinates of the robot after the execution of the above motion command are given by $\mathcal{T}_1(R_1 \cap R_2) \cup \mathcal{T}_2(R_1 \cap R_2)$.

Let us now consider the case of confusable landmarks that are being intersected by the omnidirectional preimage of some extension set. Two confusable landmarks will become part of the next goal extension only if they have identical perfect-control motion commands and identical imperfect-control motion commands. Assume that the robot senses its orientation independently of the landmarks (e.g., uses a compass). In this case, the commanded velocity direction, specified in the global frame of reference, must be the same for both landmarks. Thus, both landmark regions must be intersected by the same directional preimage. Additionally, the intersections of this preimage with the landmark regions must have a common region (specified in the local coordinate frames), which will become the perfect-control motion command (see Figure 5.10).

If the robot computes its orientation relative to the landmarks, then the commanded direction of motion is specified in the landmark coordinate systems. This situation is trickier. Since the robot cannot distinguish L_1 from L_2, the specified

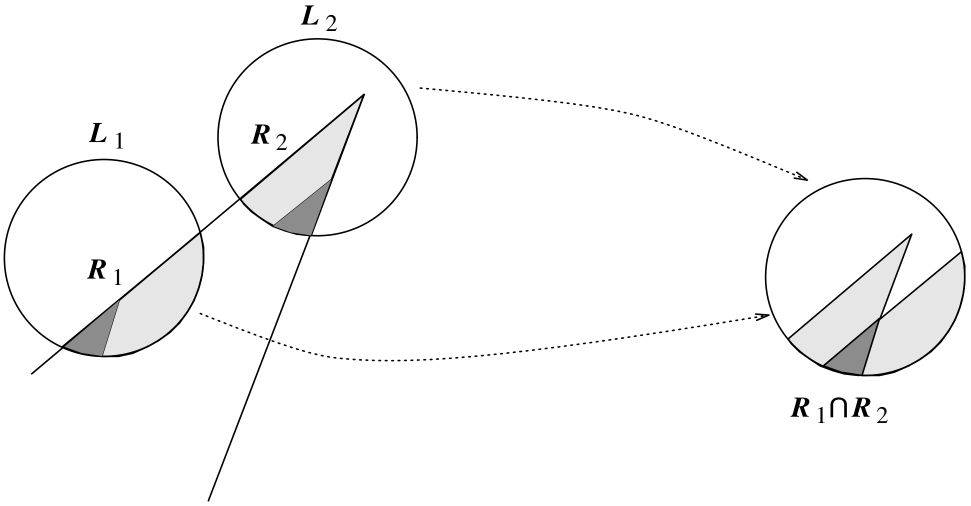


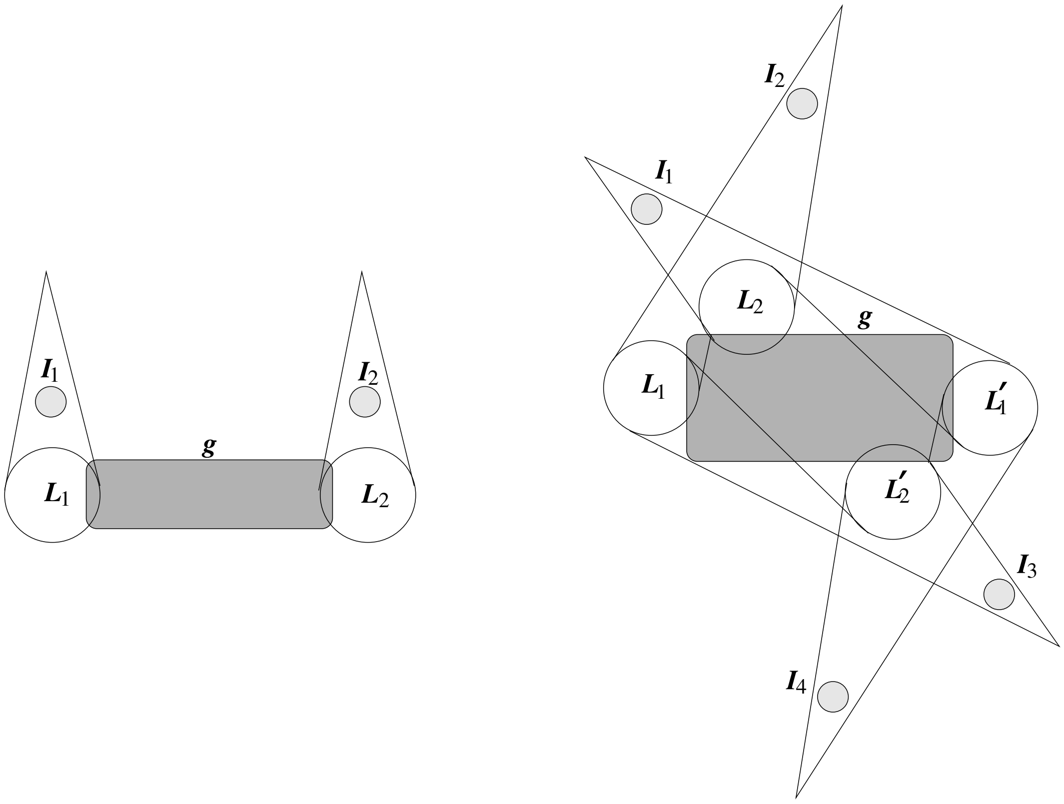
Figure 5.10: Preimage intersecting confusable landmarks

(relative) value of the commanded direction d must be the same for both disks. Let α be the absolute orientation of the frame of reference attached to L_1, and β the orientation of the frame of reference attached to L_2. If the robot is in L_1, its absolute commanded direction will be $\alpha + d$; if it is in L_2, its absolute commanded direction will be $\beta + d$. If $\alpha \neq \beta$, we cannot use a single directional preimage to intersect both landmark disks. If $\mathcal{E}$ represents the current extension set, we need to find an appropriate value of d, such that, if $R_1 = \mathcal{P}(d+\alpha; \mathcal{E}) \cap L_1$ and $R_2 = \mathcal{P}(d+\beta; \mathcal{E}) \cap L_2$, with R_1 and R_2 specified in the local frames of reference attached to L_1 and L_2 respectively, then $R_1 \cap R_2 \neq \emptyset$.

5.6.2 When no common strategy exists

If it is not possible to find a common (relative) motion command for two confusable landmark disks, we may not include them both in the same extension set. Instead, we add them one at a time, considering all possible cases. Each time, the landmark disk that does not become part of the extension set must be considered as a forbidden region (an obstacle). Otherwise, it can be mistaken for the landmark disk that belongs to the kernel, and cause premature termination of the motion command.

It is necessary to try all possible kernels, because otherwise the completeness of

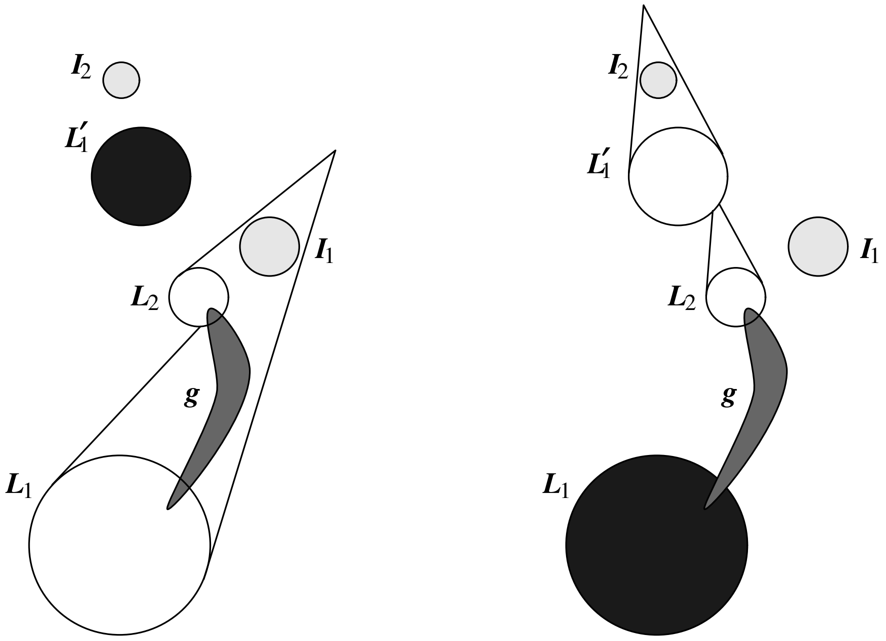


(a) (b)

Figure 5.11: Planning with confusable landmarks

the planner would be in jeopardy. For example, in Figure 5.11(a) the confusable landmarks L_1 and L_2 are intersected by the goal but have no common intersection region. Extension (L_1, L_2) cannot be considered, because if the initial region were $I_1 \cup I_2$, then the robot could enter either L_1 or L_2, and it wouldn't know which way to move in order to reach the goal. If the initial region is I_2 we will miss an existing plan if we only consider extension set (L_1); if the initial region is I_1 we will miss an existing plan if we only consider extension set (L_2). Therefore, we need to consider both combinations: L_1 in the extension set with L_2 being an obstacle, and L_2 in the extension set with L_1 being an obstacle.

When there are more confusable landmarks, things get more complicated. In Figure 5.11(b), L_1 is confusable with L'_1 and L_2 is confusable with L'_2. All of them intersect the goal. We have to check all possible combinations of extension sets,



(a) (b)

Figure 5.12: Considering all possible extension sets

(L_1, L_2), (L_1, L_2'), (L_1', L_2) and (L_1', L_2'), with the remaining disks being treated as obstacles each time. In this figure, it is evident that for each possible extension set there exists a location of an initial region disk, for which this particular extension set can produce a plan, but no other possible extension set can do so. For example, in Figure 5.11(b) I_2 is included in the preimage of (L_1, L_2) with L_1' and L_2' being treated as forbidden regions. No other valid extension set can produce a preimage that includes I_2.

In general, if n pairs of confusable disks (with no common region) have to be considered for the next extension set, then all 2^n possible combinations of non-confusable landmarks must be tried. In this case, a complete algorithm is exponential.

5.6.3 More complications

Even when the extension set contains no two landmarks that are confusable, we still have to be careful. If there are non-extension landmarks that are confusable with some landmark in the extension set, they must be considered forbidden regions, because they may cause the robot to erroneously believe that it has achieved the extension set. Therefore, the effect of adding a landmark disk to the extension set may not be monotonic with respect to the existence of a plan, because it may introduce forbidden regions that make the problem harder. As a result, it is not sufficient to consider the biggest possible extension set when looking for a plan.

Consider the example of Figure 5.12. The goal $\mathcal{G}$ intersects the non-confusable landmark disks L_1 and L_2. A third landmark L_1', confusable with L_1, exists in the workspace. Normally, we add both L_1 and L_2 in the extension set (Figure 5.12(a)). This makes L_1' a forbidden region, so it is depicted in dark grey. If the initial region is disk I_1, a guaranteed plan exists. However, if the initial region is disk I_2, no plan can be found, because I_2 is hidden behind the forbidden region L_1'. On the other hand, if we keep L_1 outside the extension set, the preimage of L_2 intersects L_1', and then the preimage of both of them (with L_1 being a forbidden region) includes I_2 but not I_1 (Figure 5.12(b)). The first choice of the extension set returns a plan for $\mathcal{I} = \{I_1\}$, but no plan for $\mathcal{I} = \{I_2\}$. The inverse is true for the second choice. Consequently, a complete planner must consider both cases.

Let $\mathcal{E}$ be a potential maximal extension set that does not contain any pair of confusable landmarks. Let $\mathcal{F}$ be a subset of $\mathcal{E}$ containing all landmarks of $\mathcal{E}$ that may be confused with some landmark outside $\mathcal{E}$. A complete planner needs to consider all possible extension sets $(\mathcal{E} \setminus \mathcal{F}) \cup \mathcal{S}$, where $\mathcal{S}$ is any subset of $\mathcal{F}$. Once again, this is an exponential number of extension sets.

It is evident that the existence of confusable landmarks makes the planning problem significantly harder. Although algorithms that deal with this case can be found, it may be preferable to avoid confusable landmarks altogether. After all, this is the basic idea of this thesis: Find the factors that contribute the most to the complexity of the planning with uncertainty problem, and engineer the robot and the workspace, so that these factors do not affect the problem any more. Since confusable landmarks make the planning problem intractable, we must try to use only distinguishable landmarks.

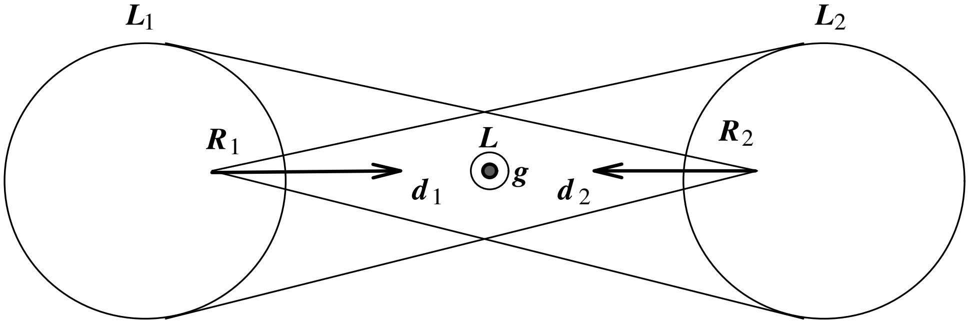


Figure 5.13: A stochastic plan

5.7 Stochastic plans

Up to now we have examined only plans which are guaranteed to succeed after a bounded number of steps. In real life the class of planning problems that admits such solutions may be too limited. Imagine, for example, trying to put a key in a lock without looking. First, the key is moved to touch the lock. Then it is moved to and fro until the slot is found. The termination of the to and fro motions is based on the detection of the edge of the lock area. The number of such motions may be unbounded, but usually it takes only a few of them until the key slides into the slot. The expected duration of this plan (in steps) is bounded (see [34]).

A simpler formulation of the above example using landmarks can be found in Figure 5.13. There are three landmarks, two big ones L_1 and L_2, and a small one L between them. The goal intersects only L. The omnidirectional preimage of L does not intersect either L_1 or L_2. But a preimage of L_1 and L intersects L_2, and a preimage of L_2 and L intersects L_1. Let $R_1 = \mathcal{P}(d_1; \{L_2, L\}) \cap L_1$ and $R_2 = \mathcal{P}(d_2; \{L_1, L\}) \cap L_2$. A robot starting from R_1 and following d_1 has some probability p to hit L (and from there go to the goal) and probability $q = 1 - p$ to terminate its motion on L_2. In the latter case, it can move into R_2 and follow d_2. Let us assume that everything is symmetric, so that the probability that it stops in L is again p and the probability that it goes back to L_1 is q. In the worst case, the robot may move between L_1 and L_2 for ever. However, the expected number of steps before the robot gets into the goal can be easily found to be $1/p$, which is a finite number if p is positive.

Stochastic plans like the above have been investigated in detail in [34, 43], but the notion of landmarks facilitates their study. A deterministic plan, like the ones we have been producing up to now, can be represented as a directed acyclic graph with the following properties. One node represents the initial position, one node represents the goal, and all other nodes represent landmark disks. There is one motion command attached to each node except for the goal node. Edges stemming from each landmark and initial position node $\mathbf{n}_1$ connect it with each landmark or goal node $\mathbf{n}_2$ which the robot may reach, if it starts from $\mathbf{n}_1$ and executes the motion command attached to it. If $\mathbf{i}$ is the initial position node and $\mathbf{g}$ is the goal node, then the existence of a path can be verified by finding the truth value of the following recursive function:

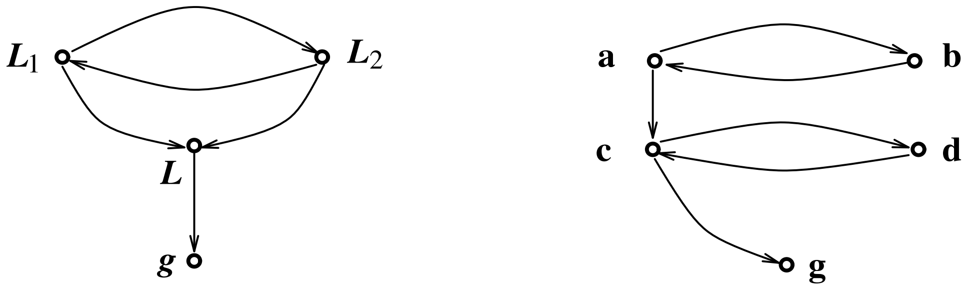
$$path(\mathbf{i},\mathbf{g}) \equiv (\mathbf{i}=\mathbf{g}) \vee (\exists \mathbf{mc}\ , \forall \mathbf{x} \in suc(\mathbf{i},\mathbf{mc})\ : path(\mathbf{x},\mathbf{g})),$$

where the function $suc(\mathbf{n},\mathbf{mc})$ returns the set of nodes that are achievable by a robot starting from node $\mathbf{n}$ and executing motion command $\mathbf{mc}$. The fact that the graph does not contain any cycles guarantees that the computation of the above function can be performed with a bounded number of recursions.

The basic idea behind a stochastic plan is that it allows cycles during its execution. However, the probability of achieving the goal node from any node that is reachable by the robot must be strictly positive. Otherwise, the achievement of the goal would not be probabilistically guaranteed. The same function *path* theoretically defines the existence of a stochastic path between $\mathbf{i}$ and $\mathbf{g}$. Nevertheless, since the graph in this case may contain cycles, the computation of *path* may enter an infinite loop. An alternative computable definition is the following:

$$\begin{array}{rcl}
path(\mathbf{i},\mathbf{g}) & \equiv & goodpath(\mathbf{i},\mathbf{g},nil) \\
goodpath(\mathbf{i},\mathbf{g},\mathbf{N}) & \equiv & (\mathbf{i}=\mathbf{g}) \vee \\
 & & (\mathbf{i} \notin \mathbf{N} \wedge \exists \mathbf{mc}\ , \forall \mathbf{x} \in suc(\mathbf{i},\mathbf{mc})\ : goodpath(\mathbf{x},\mathbf{g},app(\mathbf{i},\mathbf{N}))) \vee \\
 & & (\mathbf{i} \in \mathbf{N} \wedge \exists \mathbf{mc}\ , \exists \mathbf{x} \in tail(\mathbf{i},\mathbf{N})\ : goalpath(\mathbf{x},\mathbf{g},tail(\mathbf{i},\mathbf{N}))) \\
goalpath(\mathbf{i},\mathbf{g},\mathbf{N}) & \equiv & (\mathbf{i}=\mathbf{g}) \vee \\
 & & (\exists \mathbf{mc}\ , \exists \mathbf{x} \in suc(\mathbf{i},\mathbf{mc})\ : \mathbf{x} \notin \mathbf{N} \wedge goalpath(\mathbf{x},\mathbf{g},app(\mathbf{x},\mathbf{N}))).
\end{array}$$

The function $app(\mathbf{x},\mathbf{N})$ puts element $\mathbf{x}$ at the end of list $\mathbf{N}$; the function $tail(\mathbf{x},\mathbf{N})$ returns the part of $\mathbf{N}$ beginning with element $\mathbf{x}$ up to the end of the list. Starting



(a) (b)

Figure 5.14: Finding stochastic plans

from a particular node, the *goodpath* function requires that all possible successors of the node have the same "goodpath" property. It keeps track of the visited trace of nodes, and it terminates either by reaching the goal node, or by discovering a cycle. In the first case, it returns success; in the second, it attempts to find out whether there is positive probability to exit the cycle and get into the goal. This is the function *goalpath*, which terminates successfully if it reaches the goal, and returns failure if it leads back into the cycle or it creates a dead-end cycle itself.

Figure 5.14 shows two directed cyclic graphs. The first one, 5.14(a), corresponds to the example of Figure 5.13. Function $path(L_1, \mathcal{G})$ works as follows:

$$\begin{aligned}
path(L_1, \mathcal{G}) &\equiv goodpath(L_1, \mathcal{G}, nil) \\
&\equiv goodpath(L_2, \mathcal{G}, [L_1]) \wedge goodpath(L, \mathcal{G}, [L_1]) \\
&\equiv goodpath(L_1, \mathcal{G}, [L_1, L_2]) \wedge goodpath(\mathcal{G}, \mathcal{G}, [L_1, L]) \\
&\equiv (goalpath(L_1, \mathcal{G}, [L_1, L_2]) \vee goalpath(L_2, \mathcal{G}, [L_1, L_2])) \wedge 1 \\
&\equiv goalpath(L, \mathcal{G}, [L_1, L_2, L]) \vee goalpath(L, \mathcal{G}, [L_1, L_2, L]) \\
&\equiv goalpath(\mathcal{G}, \mathcal{G}, [L_1, L_2, L, \mathcal{G}]) \\
&\equiv 1.
\end{aligned}$$

Figure 5.14(b) shows a different example with two cycles:

$$\begin{aligned}
path(\mathbf{a}, \mathbf{g}) &\equiv goodpath(\mathbf{a}, \mathbf{g}, nil) \\
&\equiv goodpath(\mathbf{b}, \mathbf{g}, [\mathbf{a}]) \wedge goodpath(\mathbf{c}, \mathbf{g}, [\mathbf{a}]) \\
&\equiv goodpath(\mathbf{a}, \mathbf{g}, [\mathbf{a}, \mathbf{b}]) \wedge goodpath(\mathbf{d}, \mathbf{g}, [\mathbf{a}, \mathbf{c}]) \wedge goodpath(\mathbf{g}, \mathbf{g}, [\mathbf{a}, \mathbf{c}]) \\
&\equiv goalpath(\mathbf{a}, \mathbf{g}, [\mathbf{a}, \mathbf{b}]) \wedge goodpath(\mathbf{c}, \mathbf{g}, [\mathbf{a}, \mathbf{c}, \mathbf{d}]) \wedge 1
\end{aligned}$$

$$\begin{aligned}
&\equiv \; goalpath(\mathbf{c}, \mathbf{g}, [\mathbf{a}, \mathbf{b}, \mathbf{c}]) \wedge goalpath(\mathbf{c}, \mathbf{g}, [\mathbf{c}, \mathbf{d}]) \\
&\equiv \; goalpath(\mathbf{g}, \mathbf{g}, [\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{g}]) \wedge goalpath(\mathbf{g}, \mathbf{g}, [\mathbf{c}, \mathbf{d}, \mathbf{g}]) \\
&\equiv \; 1 \wedge 1 \equiv 1.
\end{aligned}$$

The above examples have taken a fixed graph and attempted to find a probabilistically guaranteed path on it. In reality, the topology of the graph depends on the various choices of the motion commands **mc**. Although we may consider only a finite number of motion commands (because we can group together all commands that produce a graph with the same topology), it is not clear how we can avoid considering an exponential number of them. Another interesting problem is to find the optimum combination of motion commands, so that the expected length of the execution trace of a plan is minimum. In the next chapter we discuss a method that may be used to address this problem.

Chapter 6

Multiparametric Planning

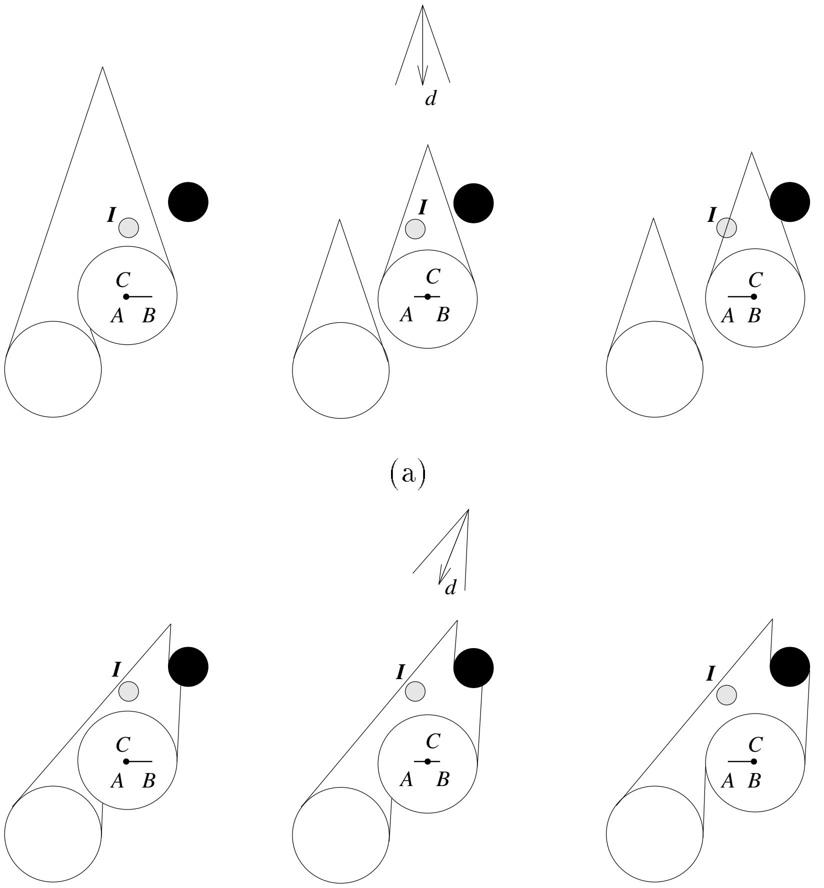
From our experimentation with the planner of Chapter 4, both in simulation and with a real robot, it became apparent that the performance of the planner depends heavily on the choice of the directional uncertainty θ. On one hand, we wish to be conservative on the choice of θ (i.e., use a big value), so that the assumption of guaranteed navigation within the control uncertainty cone is not violated. On the other hand, unnecessarily high values of θ not only cause the planner to return failure (although a plan may exist), but they also increase planning time. Furthermore, the choice of θ is not straightforward, and requires experimentation. Many times, it is possible to change the actual value of the directional uncertainty both at design time (use more accurate sensors, make the workspace floor flatter) and at execution time (make the robot go slower, tighten the motion control loop, etc.). Also, when a planner returns failure, it would be nice to know the maximum possible uncertainty θ_m for which a guaranteed plan exists. If θ_m is close enough to the actual uncertainty of the robot, then the plan found for θ_m may be usable; it is not guaranteed, but its probability of success may be quite high.

In this chapter we present a planner that computes the dependence of plans on the choice of θ. This planner can solve all the problems presented above and, as we will see, many more. The planner uses a technique for computing sufficient representations of preimages depending on more than one parameters, with the help of an algorithm that considers critical events in order to discretize the parameter space into a polynomial number of equivalence cells. Then, it suffices to compute just one preimage per cell.

6.1 Higher dimension preimages

A directional preimage $\mathcal{P}$ is a two-dimensional subset of $\Re^2$, which depends on the following parameters: the commanded direction of motion d, the directional uncertainty θ, the target set $\mathcal{T}$, the arrangement of the obstacles and landmarks in the workspace, and the control uncertainty model. In Chapter 4 we studied the dependence of the directional preimage on d when all other parameters remain constant, and we called the resulting three-dimensional set an omnidirectional preimage. The assumption that the commanded direction remains constant during the execution of a motion command enables us to describe omnidirectional preimages as a set of directional preimages computed for various values of d (slices). We have proven that a sound and complete planning algorithm needs to compute only a polynomial number of such slices.

This technique can be extended in order to study the dependence of the directional preimage on other parameters as well. For example, Donald [23] analyzed the effect of error model on motion planning. If error model can be parametrized with a single parameter, then a finite number of directional preimages (slices), computed for critical values of this parameter, can be used to adequately represent all possible preimages. For a plan to be guaranteed, all these preimages must include the initial region. In Figure 6.1(a) we show an example with model error. The commanded direction of motion d is fixed. The location of the center C of the upper landmark disk is not perfectly known, but it may lie anywhere on a small line segment AB. Let p represent the actual distance between A and C. When $p = 0$, the preimage consists of a single component that includes the initial region $\mathcal{I}$. As p grows, the topology of the preimage does not change, up to the point where the preimage breaks into two components. At this point $\mathcal{I}$ is still included in the preimage. As p continues to grow, the left ray of the upper component moves to the right, and eventually it becomes tangent to the single disk in $\mathcal{I}$. From that point on, the initial region stops being included in the preimage. The topology of the preimage changes once more, when the right ray of



(a)

(b)

Figure 6.1: A two-parametric motion planning problem

the upper component hits the obstacle. Topological changes of the preimage occur at critical values of p, for which an edge of the boundary of the preimage becomes tangent to a landmark or obstacle disk. The inclusion of $\mathcal{I}$ in the preimage may change at critical events that occur when an edge of the boundary of the preimage becomes tangent to a disk in $\mathcal{I}$. In the example of Figure 6.1(a) $\mathcal{I}$ is not included in the preimage for all possible values of p, therefore a motion command along d is not guaranteed to lead the robot into one of the two landmarks.

The above analysis is very similar to the analysis of Chapter 4. The problem becomes more interesting if we allow d to vary as well, and attempt to find one value of

d for which motion from $\mathcal{I}$ along d is guaranteed to terminate on one of the landmarks despite control uncertainty and model error. In Figure 6.1(b) we show such a value of d. It has been selected so that the left ray of the lower landmark leaves $\mathcal{I}$ to its right, and so that the breakup of the components does not occur even when C reaches its rightmost location. A direction d with these properties can be found as follows: First, observe that each critical event occurs when a preimage edge is tangent to a disk, or a spike of the preimage intersects a disk. These conditions occur for particular combinations of the changeable parameters d and p. The relation between d and p, for which a critical event occurs, defines a critical curve in the dp-plane. A motion across a critical curve (in dp-space) triggers a change of some property of the preimage. If we compute the curves that correspond to all critical events and overlay them, the dp-plane is divided into cells. All points in a cell lie on the same side of all curves, therefore the properties of the preimage (that are relevant to the planning problem) remain unchanged at each point of a cell. Therefore, it suffices to examine one point in each cell, in order to find a preimage with desirable properties. The number of cells is equal to the square of the number of critical events. The number of critical events is usually a polynomial function of the complexity of the workspace. Thus the overall algorithm is polynomial, too. The collection of directional preimages, each corresponding to a unique cell in the decomposition of the dp-space, is a polynomial description of a four-dimensional structure, representing the omnidirectional preimage of the goal set for each possible value of the parameter p.

In general, we can solve problems where the directional preimage of a target set $\mathcal{T}$ depends on a n-dimensional parameter $\mathbf{p} = (p_1, p_2, \ldots, p_n) \in \mathcal{D}_{\mathbf{p}}$ (with the commanded direction d being one of the p_i's). In this case, critical events correspond to hyperplanes in the n-dimensional domain of $\mathbf{p}$. These hyperplanes subdivide $\mathcal{D}_{\mathbf{p}}$ into cells. The $n+1$-dimensional preimage can be then represented by a finite number of 2-dimensional slices (directional preimages). This number is equal to the number of cells in the subdivision.

6.2 Planning with controllable directional uncertainty

In this section we apply the above ideas to the case where $p \equiv \theta$, i.e. the changeable parameter is the directional uncertainty of the robot. In particular we assume that the robot has control over its directional uncertainty and it can adjust it to take any value in a given interval $[\theta_{min}, \theta_{max}]$. Such control is possible through other variables like the speed of motion of the robot (a fast-moving robot will have bigger directional uncertainty), or the computing resources dedicated to motion control. Undoubtedly, uncertainty cannot be shrunk for free, the robot will have to pay some kind of cost. If speed is the control variable, then the cost is the time it takes to execute the plan; if the amount of computing resources for motion control determines the uncertainty, then the cost is the opportunity cost of not using these resources for some other function.

When both d and θ change, the directional preimage of a set of landmark regions changes by deformation, except for certain combinations of values of d and θ where topological changes occur. All pairs of d and θ for which the same topological change occurs form a *critical curve* in the $d\theta$-space. The four-dimensional preimage corresponding to all possible uncertainty values and all possible velocity directions can be represented by a polynomial number of 2-dimensional slices (directional preimages).

Using this representation we develop two basic planners, a one-step planner and a multi-step planner. The former finds the combination of d and θ for which a motion command along d with uncertainty θ is guaranteed to lead the robot into the goal, and at the same time the cost paid for the choice of θ is minimum. The second uses a greedy algorithm to create multi-step guaranteed plans with minimal cost at each step but not necessarily minimal overall cost.

The same planners can be used even when the directional uncertainty is not controllable by the robot, but either varies on its own (anisotropic workspaces like rivers), or is actually constant. In the latter case, analyzing the dependence of plans on θ allows us to develop more robust plans (so that the robot may react to unexpected events at execution time without the need for replanning), or, when no guaranteed plan exists, to find non-guaranteed plans with maximum likelihood of success.

Finally, we mention several other possible applications of two-parametric planning problems that can be solved with similar algorithms. The only requirement is that the number of critical curves remains polynomial, and that the equations of critical curves can be easily computed and solved.

6.2.1 Statement of the problem

The problem we attempt to solve in this chapter differs from the problem of Chapter 4 only in the definition of the imperfect-control mode motion command (I-command). Therefore, the problem statement of Section 4.2 can be used to define this problem as well, with just the following exception: Instead of defining an I-command as a pair of a commanded velocity and a termination set, we now define it as a triplet $(d, \theta, \mathcal{L})$, where $d \in S^1$ is the commanded direction of motion, $\theta \in [\theta_{min}, \theta_{max}]$ is the directional uncertainty, and $\mathcal{L}$ is a set of landmark disks defining the termination set (the robot stops as soon as it enters one of these disks). The directional uncertainty is now part of the imperfect motion command, because we assume that it is controllable by the robot in a connected interval $[\theta_{min}, \theta_{max}] \subset [0, \pi/2)$. A decreasing function $c(\theta)$ defines the cost of navigating with uncertainty θ.

We will use ℓ to describe the number of landmark disks in the workspace. The number of obstacles is in $O(\ell)$. We will denote with s the number of maximal connected sets of landmark disks (landmark areas). Given a goal region $\mathcal{G}$ and an initial region $\mathcal{I}$, the problem is to generate a sequence of motion commands to make the robot move from its initial position in $\mathcal{I}$ into $\mathcal{G}$ and stop there. In addition, we wish to minimize the cost paid for the various choices of directional uncertainty in the imperfect-control motion commands.

6.2.2 Outline of a planning algorithm

The planning method we use follows the technique outlined in Subsection 4.3.2. We first construct the extension of the goal $\mathcal{E}(\mathcal{G})$ as the union of all landmark areas which intersect the goal. The remaining landmark disks are called the *intermediate-goal disks.* If the goal does not intersect any landmark disk, then it is considered unachievable, since the robot cannot sense its achievement. If the initial region $\mathcal{I}$ lies entirely in $\mathcal{E}(\mathcal{G})$, no further planning is needed since a correct plan to achieve the goal has already been found; the robot can move into the goal with a single perfect-control motion.

The preimage of $\mathcal{E}(\mathcal{G})$ for the pair (d, θ) is the maximal set of points, such that executing the imperfect-control motion command $(d, \theta, \mathcal{E}(\mathcal{G}))$ from any of these points is guaranteed to reach $\mathcal{E}(\mathcal{G})$. By definition of the goal extension, the robot cannot move into it using the perfect-control mode. Therefore, if the initial region is not a subset of the goal extension, the planner must try to find a pair (d, θ), such that the initial region $\mathcal{I}$ is contained in the preimage of $\mathcal{E}(\mathcal{G})$ for (d, θ). If one such pair (d, θ) is found, the command $(d, \theta, \mathcal{E}(\mathcal{G}))$ starting from anywhere within $\mathcal{I}$ is guaranteed to attain and terminate in $\mathcal{E}(\mathcal{G})$. From there a perfect-control motion command will achieve the goal $\mathcal{G}$. We call this plan a *one-step* plan.

A one-step plan may not exist, or may not be desirable, if its cost is too high. Then the planner can attempt to create a *multi-step* plan iteratively. At each iteration, it selects a pair (d, θ), such that the corresponding preimage of the current goal extension intersects one or more intermediate-goal disks. The disks in all landmark areas containing the intersected landmark disks are added to the goal extension, and stop being intermediate-goal disks. The new larger extension set is used for the next iteration of the planner; The preimage of the new goal extension is computed, and so on, until the initial region is contained in a preimage, or no new intermediate-goal disks can be intersected, in which case a correct plan cannot possibly exist.

The construction of the search space explored by the above algorithm requires discretizing $S^1 \times [\theta_{min}, \theta_{max}]$, i.e., the continuous $d\theta$-space. In other words, we must answer the following question: At each iteration, which values of (d, θ) should the planner consider? In the next subsection we show that the $d\theta$-space can be decomposed into a finite number of cells and that only one pair (d, θ) need be considered in each cell to ensure that the planner is complete. From this result we derive a finite search space that can be explored exhaustively, if necessary.

Since we allow θ to vary, one may wonder if the planner can ever return failure. The answer is obviously yes if the lower bound on θ is strictly positive. It remains yes, even if θ is allowed to become zero and $\mathcal{G}$ intersects a landmark disk. For example, this happens if the workspace contains a single landmark disk L intersecting $\mathcal{G}$, while $\mathcal{I}$ consists of a disk that is larger than L.

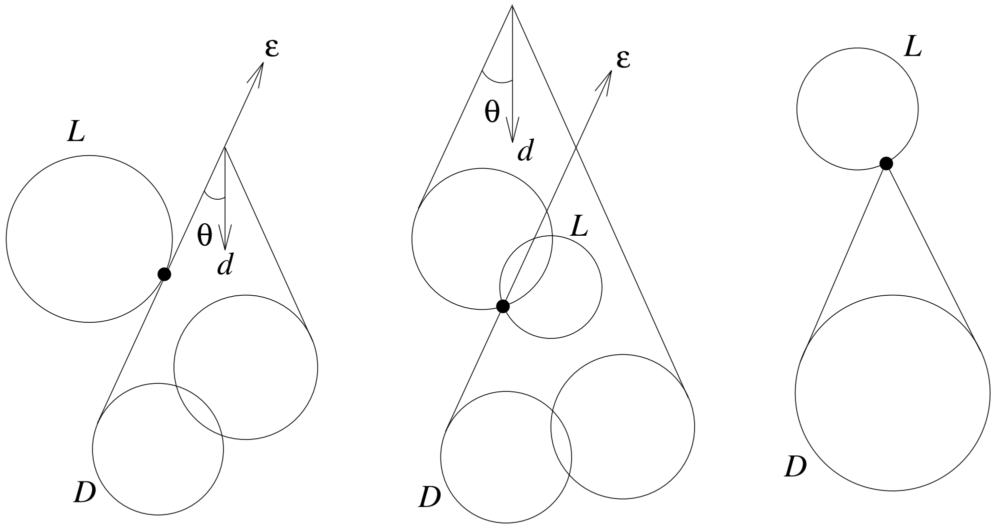
6.2.3 Building the discrete state space

Each planning iteration requires selecting a pair (d, θ) such that the preimage of the current goal extension either contains the initial region $\mathcal{I}$, or intersects intermediate-goal disks. We now show that the $d\theta$-space can be partitioned into an arrangement of *cells* of dimensions 2, 1 (line segments), and 0 (points), which are regular in the following sense: The preimage of the goal extension for any pair (d, θ) in a cell C contains the same initial-position disks and intersects the same intermediate-goal disks as the preimage for any other pair (d', θ') in C. The number of cells is polynomial in the number of landmark and obstacle disks.

Definition 6.1 (Critical points) *Given an extension set $\mathcal{E}$ in the workspace, a point (d_c, θ_c) is considered critical, if there exists another point (d'_c, θ'_c) lying arbitrarily close to (d_c, θ_c) in the $d\theta$-space, such that the preimages of $\mathcal{E}$ computed for each point have the following property: Either the sets of initial-region disks included in each preimage are not equal, or the sets of intermediate-goal disks intersected by each preimage are not equal.*

Critical points occur when an initial-region or intermediate-goal disk becomes tangent (internally or externally) to the preimage. Moreover, critical points may occur when the preimage changes discontinuously during a catastrophic event. As we discussed in Subsection 4.5.1, catastrophic events may occur only when two preimage components become tangent to each other.

Therefore a critical point corresponds to a tangency condition between a preimage component and a disk, or between two preimage components. We call such a tangency condition a *critical event.* Depending on what elements of the components' contours come in contact we distinguish *edge-to-arc* tangency, *vertex-to-arc* tangency,



(a) (b) (c)

Figure 6.2: Tangency conditions that create critical events

and *spike-to-arc* tangency. The remaining possible cases correspond to degenerate arrangements and need not be considered if we assume general positioning of objects. All the critical points that correspond to the same critical event lie on a curve in the $d\theta$-space, which we call a *critical curve*.

In Figure 6.2(a) we show an edge-to-arc tangency condition. In this case the critical curve is a straight line. Indeed, since the preimage edge that is tangent to disk L is part of a left ray that is already tangent to extension disk D, its direction must be equal to the fixed direction ε of the internal common tangent of the two disks. If d is the commanded velocity direction and θ the directional uncertainty, then ε must be equal to $d + \pi - \theta$. If the tangent edge were a part of a right ray, then ε would be equal to $d + \pi + \theta$. Therefore, the equation of a critical curve that corresponds to an edge-to-arc tangency is $d = \pm\theta + \varepsilon - \pi$, which defines a straight line with slope ± 1.

A similar critical curve corresponds to the case of vertex-to-arc tangency, shown in Figure 6.2(b). Disk L is internally tangent to a preimage component, so it lies completely inside the component. In this case a ray and two disks pass from the same point. Since the ray must pass through a fixed point (the intersection of the

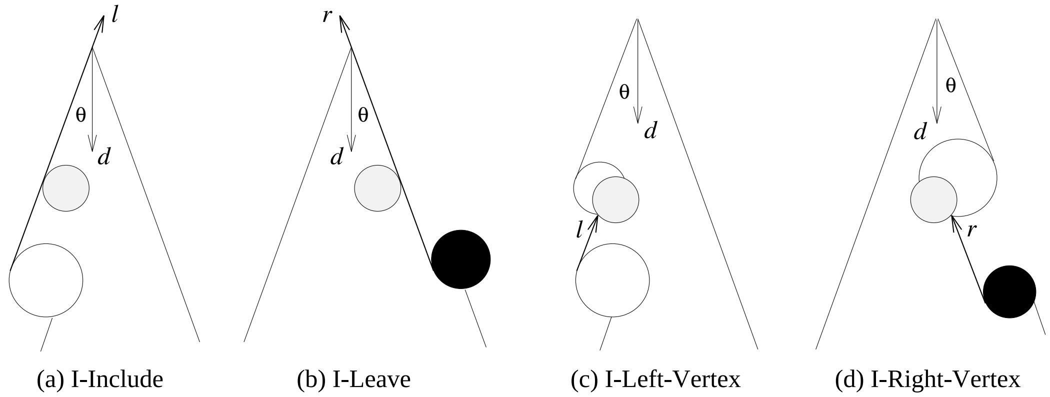


Figure 6.3: I-critical events

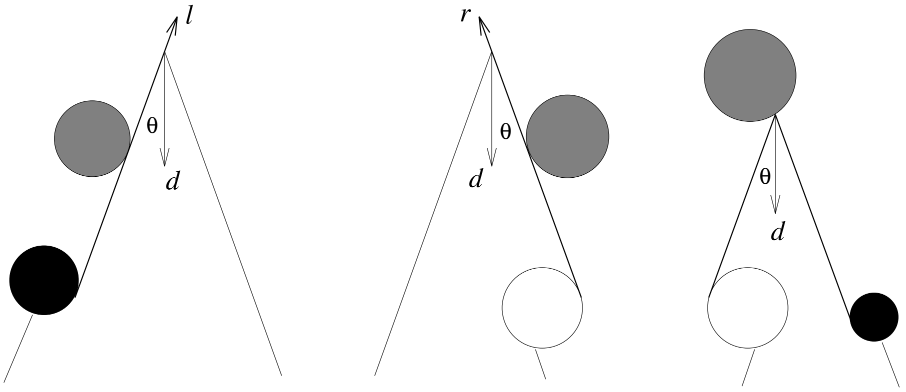
two disks), its direction must be equal to a specific value ε, defined by the direction of the left tangent from the fixed point to disk D. Therefore, the corresponding critical curve is once again a straight line with slope of ± 1.

Unfortunately, spike-to-arc intersections (Figure 6.2(c)) do not generate such simple critical curves. We call the critical curve that corresponds to this condition a *spike curve* and we represent it with $\theta = f_{spike}(d)$. In Appendix B we show that a spike curve has a single maximum and just one intersection with any line of slope ± 1. This is useful because the number of the intersections of a spike curve with the straight line critical curves is limited to one per curve.

Before even the planning search begins, it is possible to exhaustively list all (d, θ) pairs that may cause a critical event. However, it is not possible to know a priori whether a critical event will actually occur or not, without computing the preimage. Because of that, we refer to all such value pairs as *potentially critical.* We distinguish three kinds of potentially critical events, *I-critical*, *L-critical*, and *E-critical events.*

I-critical events These occur when an initial-region disk is about to be included, or stop being included in the preimage. There are four types of them, shown in Figure 6.3. All correspond to straight-line critical curves.

I-Include event : A left ray of the preimage is tangent to an initial-region disk, with this disk on its right-hand side.



(a) L-Reach (b) L-Exclude (c) L-Spike

Figure 6.4: L-critical events

I-Leave event : A right ray of the preimage is tangent to an initial-region disk, with this disk on its left.

I-Left-Vertex event : The endpoint of a left ray of the preimage coincides with the entry intersection point[1] of an initial-region disk by an extension disk.

I-Right-Vertex event : The endpoint of a right ray of the preimage coincides with the exit intersection point of an initial-region disk by an extension disk.

L-critical events These occur when an intermediate-goal disk is about to be intersected or stop being intersected by the preimage. There are three types of them, shown in Figure 6.4.

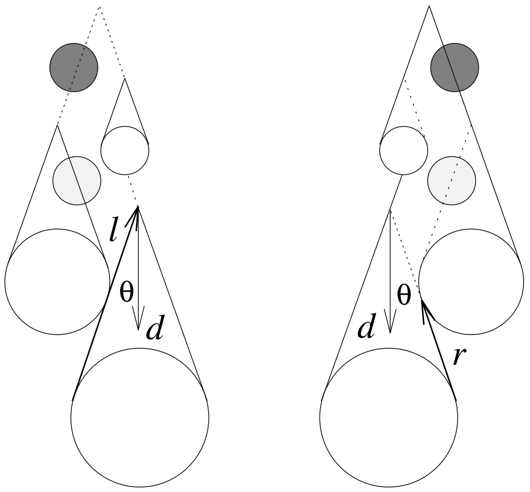
L-Reach event : A left ray of the preimage is tangent to an intermediate-goal disk, with this disk on its left.

L-Exclude event : A right ray of the preimage is tangent to an intermediate-goal disk, with this disk on its right.

L-Spike event : A spike of the preimage lies on the boundary of an intermediate-goal disk.

---

[1] When a disk δ_1 intersects another one, δ_2, we define the *entry intersection point* of δ_2 by δ_1, as the point where we enter δ_2 when we move counterclockwisely along the boundary of δ_1. The point where we exit δ_2 is the *exit intersection point*.



(a) E-Reach (b) E-Exclude

Figure 6.5: E-critical events

L-Reach and L-Exclude events have a straight line critical curve. However, the critical curve that corresponds to L-Spike events is a spike curve.

E-critical events Whereas I-critical events and L-critical events can be attributed to the interaction of a ray or a spike of the preimage with an initial-region or intermediate-goal disk, E-critical events affect primarily the topology of the preimage, which changes discontinuously. During this change, initial-region disks may get into or out of the preimage, and intermediate-goal disks may begin or stop being intersected by the preimage. There is only two E-critical events, both resulting in a straight line critical curve.

E-Reach event : A left ray of the preimage is tangent to an extension disk, with this disk on its left.

E-Exclude event : A right ray of the preimage is tangent to an extension disk, with this disk on its right.

As we mentioned before, not all the points of the curves that correspond to the above events are actually critical. However, we know that any actual critical point has to lie on one of the curves. The network of these curves determines an arrangement

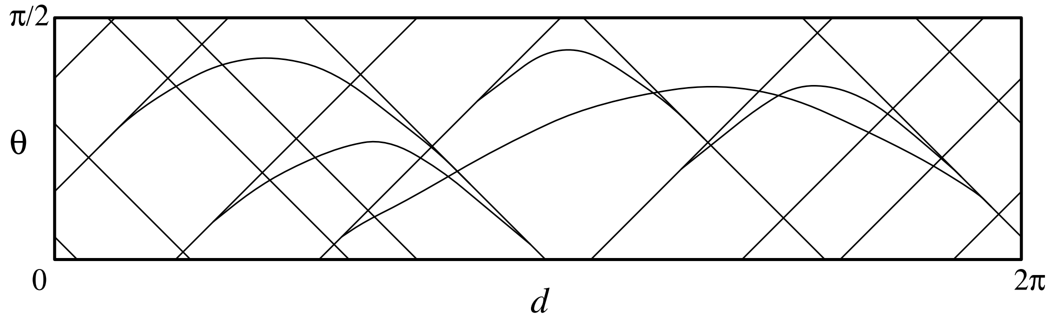


Figure 6.6: Cell arrangement in the $d\theta$-plane

of cells in the $d\theta$-space. Figure 6.6 shows a cell arrangement with $\theta \in (0, \pi/2)$. Each L-Spike curve can be attributed to a unique combination of three disks. Every other critical curve can be attributed to a combination of just two disks, and only up to a four such curves can be attributed to the same disk combination. Now, we can count the number of cells as follows:

The number of landmark disks is ℓ, the number of obstacle disks is $O(\ell)$, and the number of initial-region disks is assumed constant. Then the number of extension and intermediate-goal disks is $O(\ell)$. The number of I-Include and I-Leave curves is $O(\ell)$, the number of I-Left-Vertex, I-Right-Vertex, L-Reach, L-Exclude, E-Reach and E-Exclude curves is $O(\ell^2)$, and the number of L-Spike curves is $O(\ell^3)$. Furthermore, any two critical curves have at most one intersection (see Appendix B for proof). Therefore:

Lemma 6.2 *The number of cells in the arrangement determined by the critical curves is* $O(\ell^6)$.

6.2.4 Searching for guaranteed plans

We will now show how one can utilize the computed cell arrangement to generate one-step and multi-step guaranteed plans for a robot that can control the parameter θ within the given interval $[\theta_{min}, \theta_{max}]$. In both cases the first step of the algorithm is to find the extension $\mathcal{E}(\mathcal{G})$ of the given goal. Then, given $\mathcal{E}(\mathcal{G})$, the appropriate cell arrangement is computed.

One-step planning Let us first consider the generation of minimal-cost one-step plans. In this case, the planner only needs to find a pair (d, θ), with $\theta \in [\theta_{min}, \theta_{max}]$, such that the preimage of the goal extension $\mathcal{E}(\mathcal{G})$ for the imperfect-control motion command $(d, \theta, \mathcal{E}(\mathcal{G}))$ fully contains the initial region $\mathcal{I}$. Thus, the planner can discard the L-critical curves in the $d\theta$-plane. The remaining I- and E-event curves (all straight lines) define an arrangement of $O(\ell^4)$ cells. In every cell, the planner can select an arbitrary pair (d, θ) and compute the corresponding preimage using the $O(\ell \log \ell)$ sweep-line algorithm presented in Subsection 4.4.2. In the worst case, the planner scans all the cells. Hence, it returns a plan or declares failure in time $O(\ell^5 \log \ell)$. The resulting planner is complete.

Minimizing the decreasing cost function $c(\theta)$ requires that we find the plan with the highest possible value of θ. The value of θ for such a plan can only be θ_{max} or the θ-coordinate of the intersection of two critical lines. Otherwise, it would always be possible to move to another (d, θ) point in the same cell (i.e. without changing the inclusion state of $\mathcal{I}$) which has a bigger θ value. This observation yields the following planning algorithm: Set θ to θ_{max} and compute the preimage of $\mathcal{E}(\mathcal{G})$ for each cell of the arrangement intersecting the line $\theta = \theta_{max}$. If one preimage contains $\mathcal{I}$, return success (and the corresponding value of d). If no preimage contains $\mathcal{I}$, scan the intersections of I- and E-event lines verifying $\theta \in [\theta_{min}, \theta_{max}]$ in decreasing order of their θ-coordinates. For every intersection point (d, θ), compute the preimage of $\mathcal{E}(\mathcal{G})$. If this preimage contains $\mathcal{I}$ return success, otherwise consider the next intersection point. Return failure if all intersections with $\theta \in [\theta_{min}, \theta_{max}]$ have been considered without success.

Using a sweep-line technique to scan the intersection points, this algorithm takes output-sensitive time $O((\ell^2 + c\ell) \log \ell)$, where $c \in O(\ell^4)$ is the rank (in decreasing order) of the value of θ selected among the θ-coordinates of the $O(\ell^4)$ intersection points of the event lines.

Theorem 6.3 *Minimum-cost one-step guaranteed plans can be found with the above algorithm in* $O((\ell^2 + c\ell) \log \ell)$ *time.*

Figure 6.7 shows a simple example run with the implemented one-step planner.

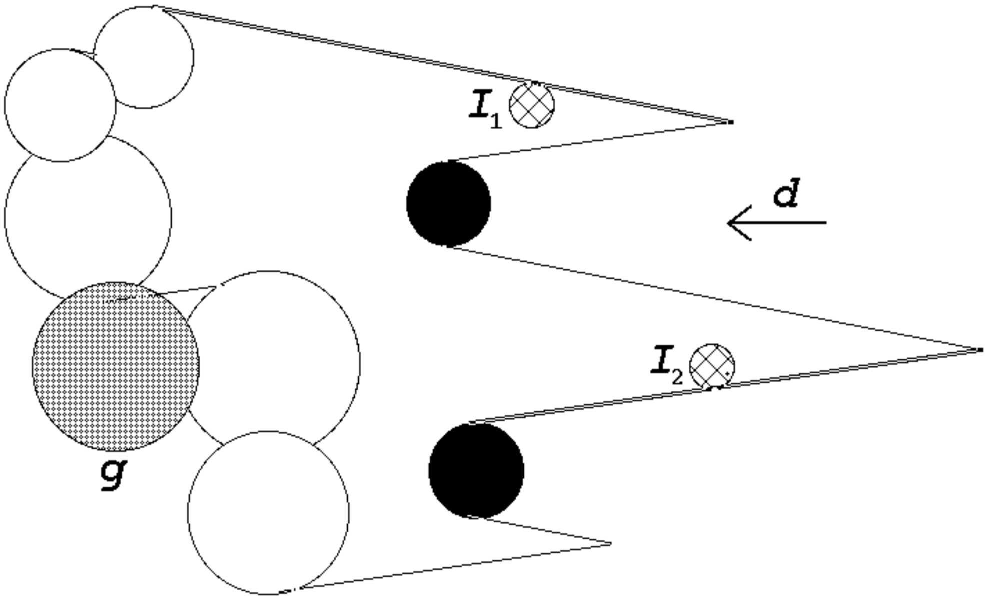


Figure 6.7: Maximum-uncertainty one-step guaranteed plan

The workspace contains two obstacle disks, and five landmark disks forming two landmark areas that both intersect the goal $\mathcal{G}$. The initial region consists of two disks, $\mathcal{I}_1$ and $\mathcal{I}_2$. The directional uncertainty is controllable in the interval [0.1,0.5] radian. The planner produces a one-step plan, i.e., a plan containing a single imperfect-control motion command $(d, \theta, \mathcal{L})$, with d shown in the figure, $\theta = 0.16$ radian, and $\mathcal{L}$ made up by the two landmark areas intersecting $\mathcal{G}$. The set of points in the workspace from which this command is guaranteed to reach $\mathcal{L}$ (i.e., the preimage of $\mathcal{L}$) is outlined in the figure. It fully contains the initial-region disks $\mathcal{I}_1$ and $\mathcal{I}_2$. The value of the control uncertainty selected by the planner is maximum over all correct one-step plans. Any slight variation of the direction of motion or increase of the directional uncertainty would cause an initial-region disk to be partially outside the preimage of $\mathcal{L}$.

Multi-step planning When the one-step strategy yields too costly a plan or no plan at all, we can perform backchaining and create multi-step plans. As explained in detail in Chapter 4, during the backchaining process we compute a sequence of successively growing extension sets. During each iteration of the planner, we add to the current extension all intermediate-goal disks that are intersected by some directional preimage of the current extension set. Backchaining stops when either we find a preimage that includes all initial-region disks, or no more intermediate-goal disks can be intersected by the preimages of the current extension set.

In order to compute minimum-cost multi-step plans, we have to generate and compare all extension set sequences that yield a guaranteed plan. However, the number of such sequences is exponential in the number of landmarks, so this algorithm would be impractical. Instead, we present a *greedy algorithm* that minimizes the cost of the current iteration step only:

1. The algorithm begins by running the fixed uncertainty planning algorithm of Chapter 4 with $\theta = \theta_{max}$. If a guaranteed plan is found, it is returned and the algorithm terminates. Otherwise, let us call $\mathcal{ME}$ the maximum extension set found (consisting of all the disks from where the robot can reliably reach the goal using the value θ_{max} for its directional uncertainty).

2. Using $\mathcal{ME}$ as the extension set and all other landmark disks as intermediate goals, we compute the cell arrangement in the $d\theta$-plane. The planner sweeps a line parallel to the d-axis in order to find the highest value of the directional uncertainty $\theta_a \in [\theta_{min}, \theta_{max}]$, such that there exists a preimage of $\mathcal{ME}$ which either contains the initial region or intersects just one intermediate-goal disk. During the sweep preimages are computed only at critical curve intersection points, as explained in the one-step planning algorithm description. However, in this case there is an additional type of points that must be visited, the points where L-critical curves reach their maximum. Indeed, below such points an intermediate-goal disk may start being intersected by the preimage.

3. If a preimage including the initial region is found, the minimum-cost multi-step plan sought is the one-step plan corresponding to that preimage.[2]

---

[2] Here, we make the assumption that the aggregate cost function has "reasonable" properties, e.g., that the cost of executing a one-step plan $[(d, \theta, \mathcal{E}(\mathcal{G}))]$ is less than the cost of executing any n-step plan $[(d_1, \theta_1, \mathcal{L}_1), \ldots, (d_n, \theta_n, \mathcal{E}(\mathcal{G}))]$, where $\theta \leq \theta_i$ for all $i \in [1, n]$.

4. If a preimage intersecting one intermediate-goal disk is found, that disk and all other disks in the same landmark area are added to $\mathcal{ME}$ and new event curves are introduced in the arrangement. Although the preimage of the new extension set may include all initial-region disks or intersect another intermediate-goal disk for a value $\theta > \theta_a$, we do not need to backtrack the sweep line (see proof below). Instead, we compute all events that correspond to intersections of the new curves (among themselves and with the old curves) that occur for values of $\theta > \theta_a$ and we sort them by decreasing θ. Before we proceed with the line-sweep, we process all these events.

5. This procedure is repeated until we find a preimage that includes all initial position disks, or no more intermediate-goal disks can be intersected. In the latter case the planner returns failure (this may happen, for example, if the initial region is too big). Since there are at most $O(\ell)$ iterations (each absorbing one intermediate-goal disk into the extension set), an output plan has ℓ steps at most.

Whenever we compute the cell arrangement, we do not need the intersections of the L-Spike event curves among themselves (unless this intersection occurs at the maximum of one or both curves), since these intersection points cannot give rise to the highest value of θ we are looking for. Indeed, if at such a point $\mathbf{c}$ (see Figure 6.8) an intermediate-goal disk is intersected by the preimage, it is always possible to move to higher values of θ while remaining beneath the curve that initiates the intermediate-goal disk intersection (one of the two intersecting L-critical curves, or some other curve that passes above $\mathbf{c}$). We do not need to consider $\mathbf{c}$, since the same intermediate-goal disk can be intersected by the preimage for a higher value of θ. As a result, we avoid checking the $O(\ell^6)$ intersection points of the spike curves with themselves, and we are left with only $O(\ell^5)$ points to test.

In order to make that possible, we need to exclude spike curves from the sweeping process (otherwise the intersections among themselves would be needed for the sweeping). All intersections of spike curves with the other curves are computed and put in the event queue, whenever a spike curve emerges at its maximum θ value. All such maximum points (whose number is $O(\ell^3)$) can be precomputed and inserted in

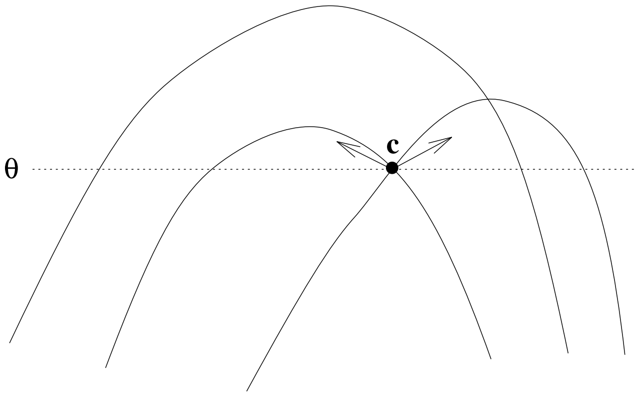


Figure 6.8: Spike curves' intersection point

the event queue before sweeping begins.

Lemma 6.4 *Each point in the sweep event queue need be visited by the planning algorithm only once.*

The proof is as follows: Assume that an intermediate-goal disk D has been just intersected by the preimage of some extension set $\mathcal{E}$ for $\theta = \theta_a$ and is being added to $\mathcal{E}$. Let us consider an already tested point (d^*, θ^*) $(\theta^* > \theta_a)$, for which the preimage of $\mathcal{E} \cup \{D\}$ intersects another intermediate-goal disk D'. (The arguments that follow hold also for the case where (d^*, θ^*) is a point for which the preimage of $\mathcal{E} \cup \{D\}$ covers the entire initial region.) A preimage consists of one or more disconnected components. Let us consider the component that intersects D'.

- **The component must include** D**.** If it consisted solely of disks in $\mathcal{E}$ this same component would have been present in the preimage of $\mathcal{E}$, when the sweep line passed from this point before. Since an intermediate-goal disk would have been intersected, the previous iteration would have been terminated at θ^*, and would not have proceeded down to θ_a.
- **The component cannot include any disks from the previous extension set** $\mathcal{E}$**.** To see this assume the component includes D and some disks from $\mathcal{E}$. If this were the case, when (d^*, θ^*) was first visited (when D was still an intermediate

goal), the preimage of the disks in $\mathcal{E}$ would have intersected D, and the iteration would have not advanced down to θ_a.

- D **cannot be the only disk of the component.** The point (d^*, θ^*) was present in the event queue before D became part of the extension set. Therefore, in general,[3] it will not be critical with respect to the intersection of D' by a preimage component that contains only D. As a result, there must exist values of θ higher than θ^*, for which this component intersects D'. This means that D' will be intersected and become part of the extension set before the new iteration proceeds down to (d^*, θ^*). During subsequent iterations the point (d^*, θ^*) will be irrelevant, as D' will not be an intermediate goal any more.

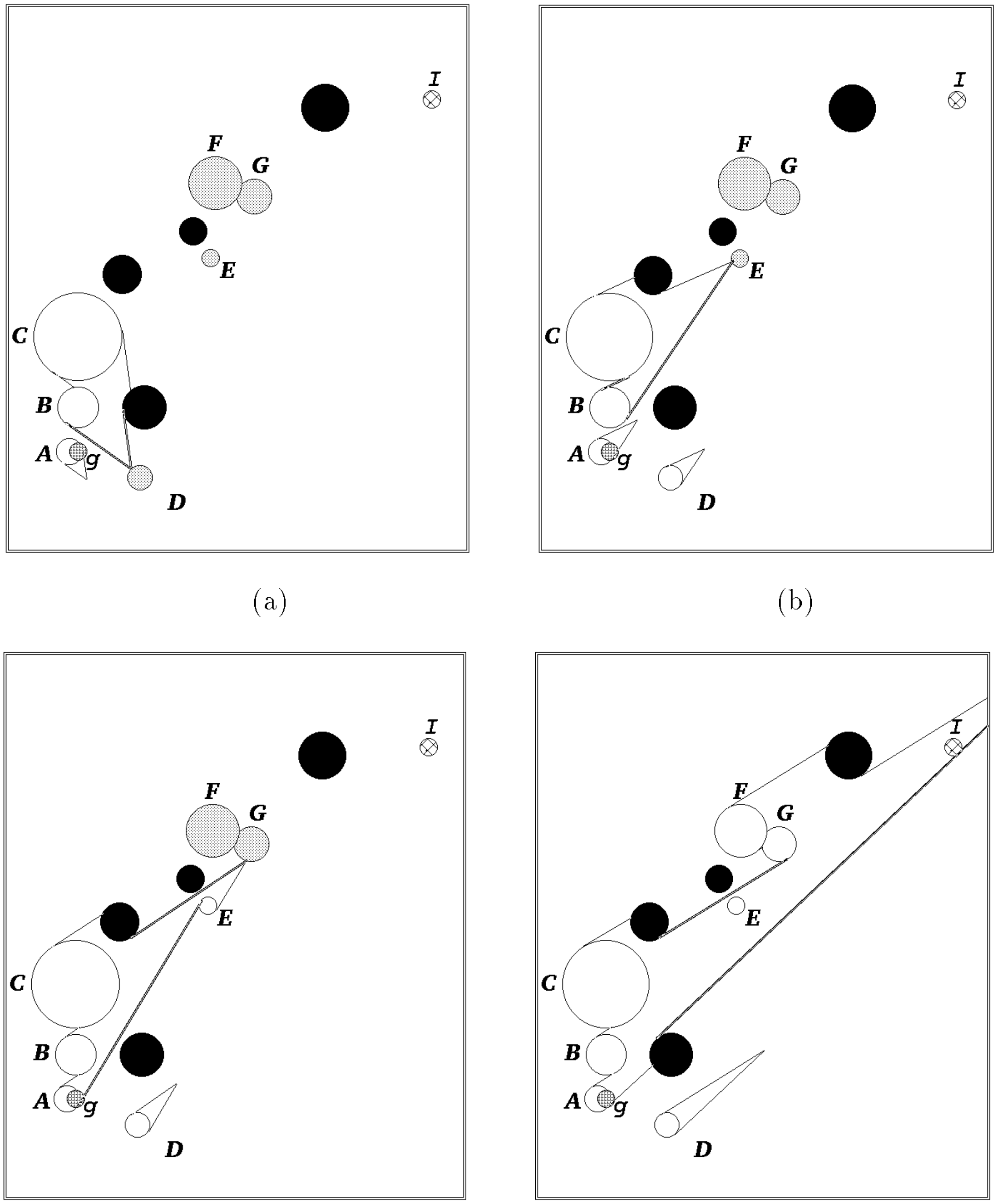
From the above arguments, it follows that if a point has already been visited once, it cannot be critical again for any extension set of subsequent iterations. As a result, we compute one directional preimage per point visited by the sweep line. Since computing the preimage takes time $O(\ell \log \ell)$ and the sweeping process visits $O(\ell^5)$ points, we derive the following theorem:

Theorem 6.5 *The total complexity of the greedy multi-step planner is* $O(\ell^6 \log \ell)$.

Figure 6.9 illustrates the operation of this algorithm with an example. The workspace contains seven landmark disks **A** through **G**, and four obstacles. The goal $\mathcal{G}$ intersects landmark disk **A**. The initial region $\mathcal{I}$ consists of a single disk. The directional uncertainty lies in the interval $[0.0, 0.5]$ radian. This means that we are not interested in uncertainty angles above 0.5 radian because the robot cannot do worse than that. Using $\theta = 0.5$, the planner first finds a guaranteed motion command to reach **A** from the landmark disks **B** and **C**. The extension of goal then consists of **A**, **B**, and **C**. For the commanded direction shown in Figure 6.9(a) and $\theta = 0.41$, the preimage of this goal extension touches **D**, which is added to the goal extension. **E** is then touched by a preimage for $\theta = 0.28$ (Figure 6.9(b)). **G** is touched by a preimage for $\theta = 0.32$ (Figure 6.9(c)). Both disks **F** and **G** are added to the goal extension, since they form a single landmark area. At this point, all landmark disks are in the

---

[3] I.e., barring a degenerate coincidence that can be disregarded.



(a) (b)

(c) (d)

Figure 6.9: Multi-step greedy planning

goal extension. The initial-position disk is then covered by a preimage for $\theta = 0.1$ (Figure 6.9(d)).

At every iteration, the algorithm maximizes directional uncertainty to achieve the current extension of the goal. In general the generated plan, if any, would not minimize a given cost function. Generating minimum-cost plans is intrinsically harder. It may require the selection of a smaller directional uncertainty at an early iteration, if this choice allows larger values of the uncertainty to be selected at subsequent iterations.

6.2.5 Extensions

The algorithms presented in this chapter were based on the same assumptions we used in Chapter 4, namely, that region shapes are circular, no contact with the obstacles is allowed, landmark and obstacle discs do not overlap or touch each other, and navigation within landmark areas is accurate.

All these assumptions can be relaxed in exactly the same manner we described in Chapter 5, without losing completeness or polynomiality. With these extensions, the presented algorithms become powerful tools for efficient planning under uncertainty.

6.3 Creating non-guaranteed plans

At the end of Chapter 4 we mentioned that there exist ways to generate non-guaranteed commands (if no guaranteed ones exist) which have better chances to succeed than a totally uninformed Brownian motion. To compute such commands we can use the same algorithm we presented in the above sections, changing only the definition of the cost function.

Consider again the problem we addressed in Chapter 4. Let the fixed directional uncertainty of the robot be θ_f. If for a given planning problem there exists no guaranteed plan, we wish to find a non-guaranteed one, whose chances to succeed are maximal.

Whenever the robot executes a motion command $(d, \mathcal{L})$, which is only guaranteed

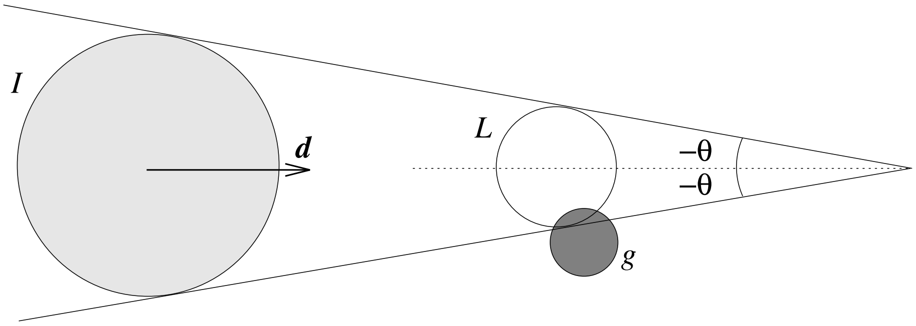


Figure 6.10: Negative directional uncertainty

to succeed for values of the directional uncertainty up to $\theta_g < \theta_f$, the robot stands the chance to miss all the landmarks in $\mathcal{L}$. Obviously, the chance of failure is a decreasing function of θ_g. Therefore, we can use the maximum value of directional uncertainty for which a motion command is guaranteed to succeed as a measure of the likelihood of success of the command. This suggests that the exact same algorithms presented in the previous sections of this chapter can be used to find such motion commands. In this case, the cost function represents the likelihood of plan failure during execution.

6.3.1 The algorithm

After planning with constant uncertainty θ_f fails, we can invoke the one-step minimum-cost planner to find the initial motion command with the highest likelihood of success. We use θ_f in place of θ_{max} (we are not interested in higher values, because we already know that they cannot lead to a guaranteed initial command), and we set $\theta_{min} = -\pi/2$. The negative values are necessary to solve the case where no guaranteed command exists even with zero directional uncertainty. This may be the case if the initial region of the robot is too big with respect to the size of the landmark areas. Figure 6.10 presents such a case. The commanded direction shown maximizes the chances of the robot to hit the landmark area.[4]

The above idea can be extended for the creation of multi-step non-guaranteed plans. Once again we set $\theta_{max} = \theta_f$ and $\theta_{min} = -\pi/2$, and we call the greedy

---

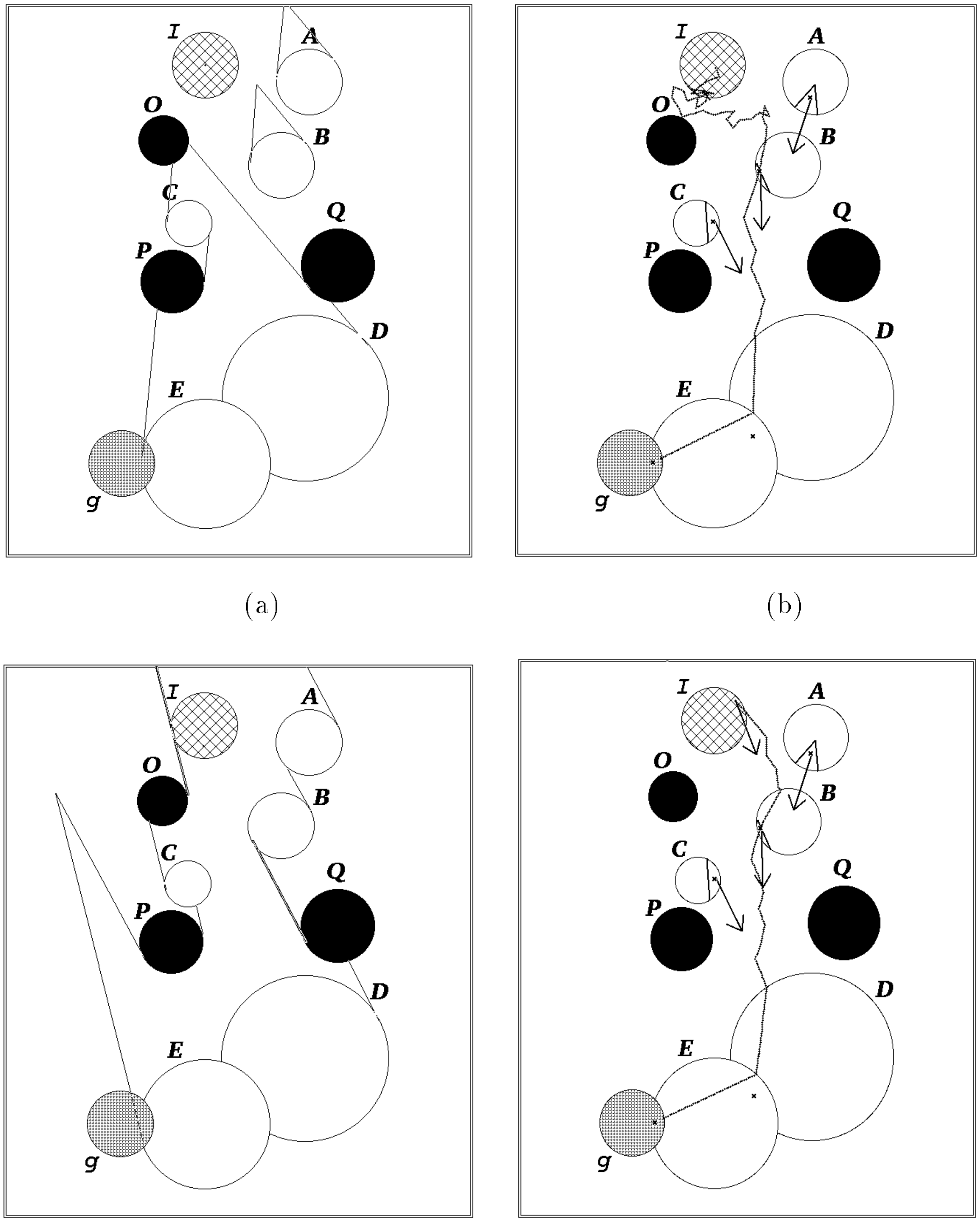
[4] Negative uncertainty may also be possible in certain environments where a potential field in the workspace forces the robot to move within valleys.

algorithm of Subsection 6.2.4. During each iteration the extension set is augmented by one landmark disk, until the initial region becomes totally included in the preimage. Note, however, that the probability of success of the total plan is not guaranteed to be maximal.

6.3.2 Implementation and examples

We implemented the above planners in a program written in the C language and running on a DEC-5000 workstation. The only major issue in this implementation concerns the computation of the maxima of the L-Spike event curves and their intersections with other event lines. We have not been able to calculate analytical expressions for these points. Therefore, our planners use traditional numerical techniques, which require some care to avoid inconsistent topological results in constructing preimages. In order to visualize the plans generated by the planners, we have developed a simple simulator. Imperfect-control motion commands are discretized into short segments and a directional error is randomly selected for each segment. The simulator allows the user to tune segment length and error distribution in the interval $[0, \theta]$, to generate various sample runs.

An example with the one-step planner In Figures 6.11(a)-(d) we present an example run with the one-step planner. In the workspace, there are five landmark disks (the white disks marked with A, B, C, D and E) and three obstacles (the black disks O, P, Q). There is one initial-region disk, marked with $\mathcal{I}$, in the upper part of the workspace, and one goal disk, marked with $\mathcal{G}$, in the lower part. In 6.11(a) we show a preimage of the extension set containing all landmark disks in the workspace calculated for $\theta = \theta_f = 0.4$ radian. It is evident that no guaranteed plan exists to achieve this goal. The planner returns an incomplete plan, so, according to the strategy presented in Section4.8, the robot executes an initial random motion with the hope that it will enter some landmark region in the extension set. In Figure 6.11(b) the robot happens to enter landmark disk B. From there, the path to the goal is guaranteed. Nevertheless, the likelihood of success of such a random motion in this example is rather low. In most attempts the robot bumps either on the obstacles, or



(a) (b)

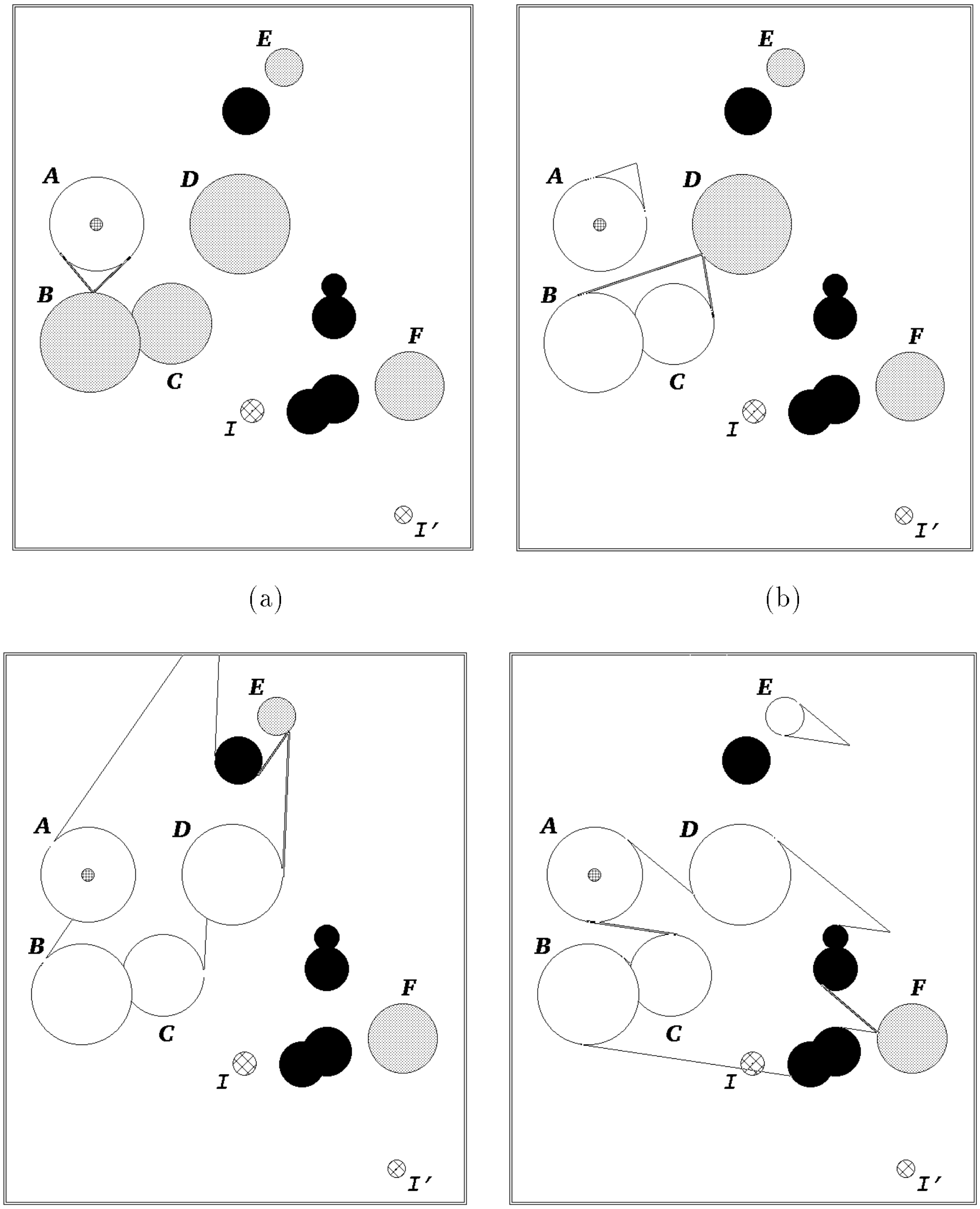
(c) (d)

Figure 6.11: Non-guaranteed plan using the one-step planner

on the boundary of the workspace.

In Figure 6.11(c) we let the planner utilize the one-step variable uncertainty planner. For $\theta = 0.12$ radian a preimage is found that includes the initial region. This value of θ is the highest value for which this inclusion occurs. If the uncertainty of the robot were 0.12 radian, then the commanded direction of motion d^* (for which the preimage in Figure 6.11(c) was computed) would allow the robot to move into a landmark disk from any point in the initial-region disk with no possibility of failure. This direction d^* is depicted by the arrow stemming from the initial region in Figure 6.11(d). It represents a non-guaranteed command: Because of its higher uncertainty, the robot might actually hit O or Q, if it tries to follow d^*. Nevertheless, this happens only rarely. This command succeeds most of the time. In Figure 6.11(d) we see such a successful execution.

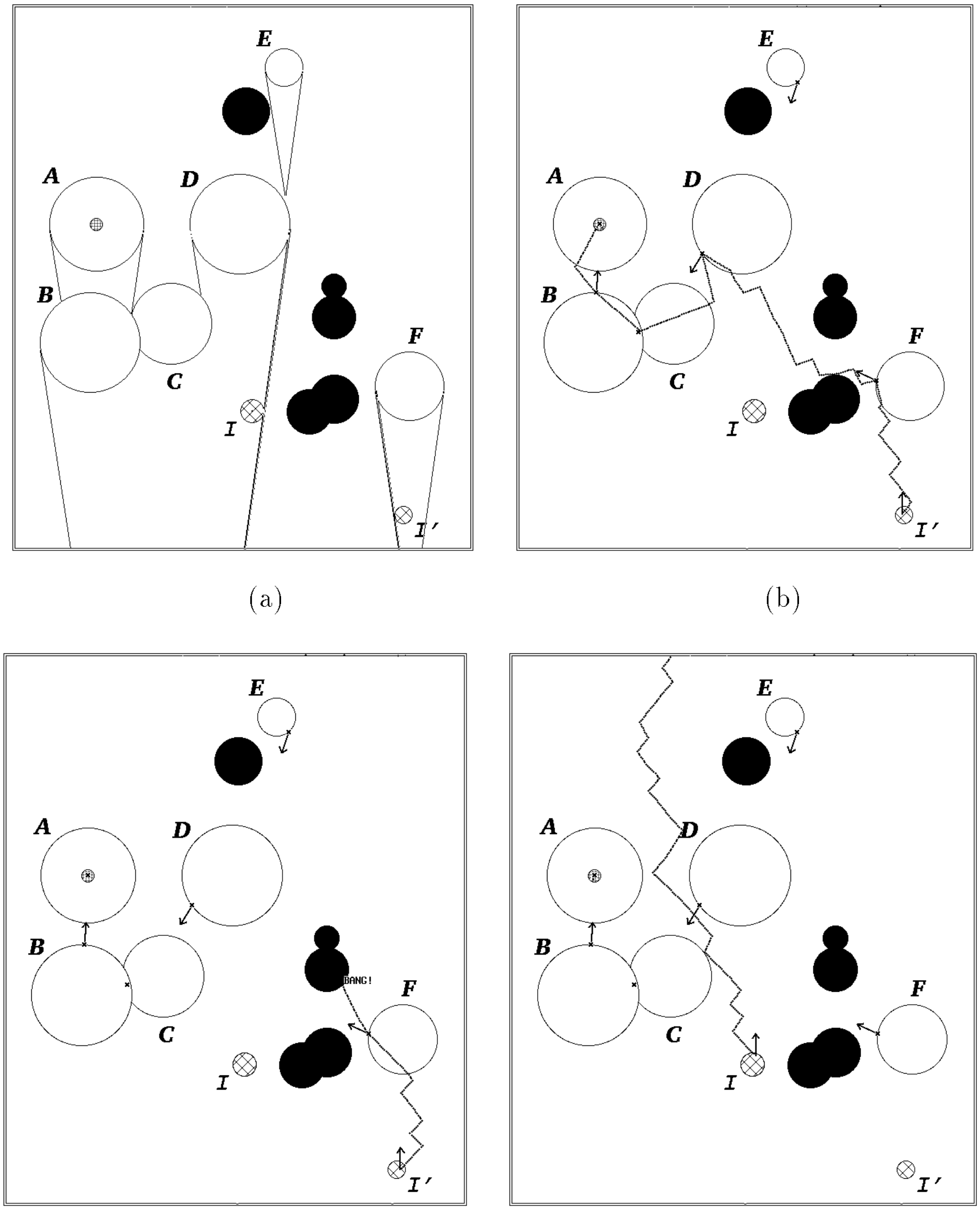
An example with the greedy multi-step planner Figures 6.12(a)-(d) and 6.13(a)-(d) present an example solved with the multi-step greedy planner. The workspace has six landmarks and five obstacles. The initial region consists of two disks, $\mathcal{I}$ and $\mathcal{I}'$. The goal region is the small disk at the center of landmark disk A. Given the nominal uncertainty of the robot $\theta_f = 0.8$ radian, the preimage of A does not intersect any intermediate-goal disk, so the constant uncertainty algorithm returns failure after one backchaining step. The greedy variable-uncertainty algorithm sweeps down to 0.75 radian, at which point disk B is intersected (Figure 6.12(a)). During the next iteration of the algorithm disk D is intersected for $\theta = 0.72$ (Figure 6.12(b)). For $\theta = 0.29$ disk E becomes part of the extension set (Figure 6.12(c)). After yet another iteration disk F is included in the extension set for $\theta = 0.27$ (Figure 6.12(d)). Finally, both initial-region disks are included in the preimage for $\theta = 0.15$ (Figure 6.13(a)). The total run time of the algorithm is 81 seconds. In Figure 6.13(b) we show a successful execution of the non-guaranteed plan. Despite its high directional uncertainty (0.8 radian), the robot finds its way to the goal safely. This is not the case though in Figure 6.13(c), where the robot crashes on an obstacle, and in Figure 6.13(d) where it hits the boundary of the workspace. After several simulated runs, the success rate of this plan was found to be 66% something



(a) (b)

(c) (d)

Figure 6.12: Example with six landmark and five obstacle disks



(a) (b)

(c) (d)

Figure 6.13: Example with six landmark and five obstacle disks (cont.)

that is rather impressive, if we take into account that the actual uncertainty of the robot is 0.8 radian. Of course, the success rate depends a lot on the way we simulate control uncertainty.[5]

In the above example, the planner sweeps 0.6 radian in one minute and 20 seconds. On the other hand, if the nominal uncertainty is set at 0.2 radian, a solution is found in about 8 seconds. With a reasonable choice of the nominal uncertainty we were able to produce plans for workspaces with up to fifty landmark and obstacle disks in a matter of a few minutes. In general, the planners' runs tend to show that the planners are more efficient than our asymptotic worst-case complexity analysis suggests. We believe that deeper combinatorial analysis of critical curves, cost amortization over iterations, and output-sensitive complexity evaluation should make it possible to produce tighter complexity results.

6.3.3 Error detection and recovery (EDR)

One major drawback of a non-guaranteed plan is that its execution may fail in a non-recognizable way. Indeed, a non-guaranteed motion may miss all the disks in its termination set and continue beyond them into uncharted territories. Introducing some awareness of time is one way to address this drawback. Another approach, inspired by Donald's EDR strategies [23], is to make sure that every non-guaranteed imperfect-control motion command inserted in the plan will either succeed or fail recognizably. Such commands have a termination set that they are guaranteed to reach, but some disks in this set are not part of the goal extension at the time they are selected.

Landmark taxonomy Consider the execution of the non-guaranteed motion command depicted in Figure 6.14. The robot moves from $\mathbf{p}$ with commanded direction of motion d and nominal directional uncertainty θ. The termination condition of the

---

[5] After the robot moves a fixed distance, it gets to choose a new actual direction of motion. In this particular implementation we had the robot choose between $d + \theta$ and $d - \theta$ with equal probability, in an attempt to bias the motion towards the edges of the uncertainty cone. However, since choices are statistically independent, most actual trajectories of the robot remain close to the axis of the uncertainty cone.

command consists of the four white disks in the figure, which form the current extension set. The grey disks are intermediate-goal landmark disks. This non-guaranteed motion command divides the workspace in three distinct sets:

The Unreachable Set: The three dark-shaded areas represent points that are unachievable by the robot with this motion command. These areas are the exterior of the forward projection of $\mathbf{p}$.[6]

The Hopeful Set: The intersection of the weak preimage of the extension set[7] with the forward projection of $\mathbf{p}$ contains all points from where the robot has positive probability to hit one of the extension set disks. It is the white area outlined with a thick border in the figure.

The Failure Set: The remaining points of the workspace can be reached by the robot, but from there there is no chance to hit one of the extension set disks. If the robot can sense that it has reached a point in this region (lightly shaded in the figure), it should stop and declare failure.

Accordingly, we can classify intermediate-goal landmarks in the following way: If they are totally included in the unreachable region we call them *unreachable landmarks*. If they intersect the hopeful region we call them *dilemma landmarks*. Finally, if they intersect the failure region but not the hopeful region, we call them *failure landmarks*. In Figure 6.14 A and B are unreachable landmarks, C and D are dilemma landmarks, and E and F are failure landmarks. Clearly, we are not interested in unreachable landmarks. On the other hand, if a robot finds itself in a failure landmark, it should stop and declare failure. From there, there may exist an already computed plan to the goal, or we may have to replan. We call this kind of failure *recognizable*. Since the failure landmarks also terminate the motion of a non-guaranteed command, they are part of the termination set of the command. The extension-set landmarks

---

[6] The forward projection of a region $\mathcal{R}$ for a given motion command $\mathbf{MC}$ consists of all points reachable by a robot starting from $\mathcal{R}$ and executing $\mathbf{MC}$. It can be computed with algorithms similar to those that compute preimages.

[7] The weak preimage of a region $\mathcal{R}$ for a given motion command $\mathbf{MC}$ consists of all points from where a robot has positive probability to reach (and stop in) $\mathcal{R}$. It can be computed with algorithms similar to those that compute preimages.

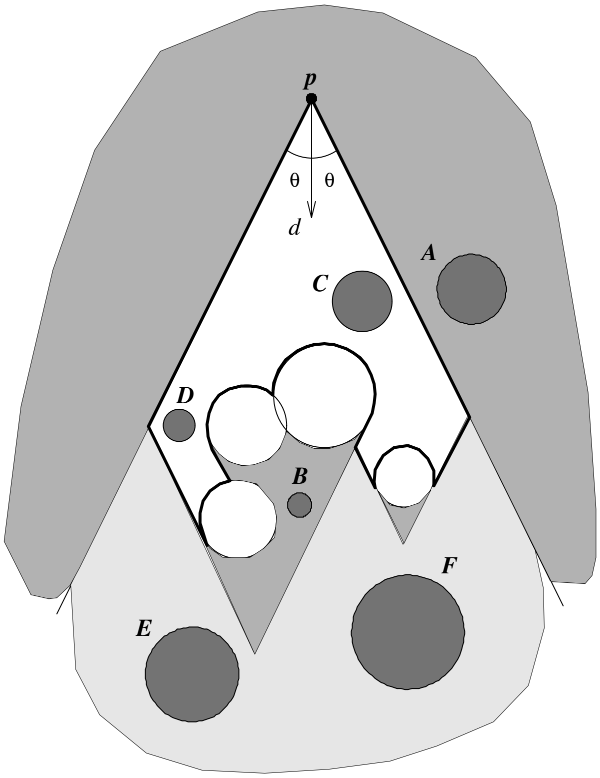


Figure 6.14: Non-guaranteed command

form the *success set* and the failure landmarks form the *failure set*. A command that either succeeds or fails recognizably is preferable to one that may fail unpredictably.

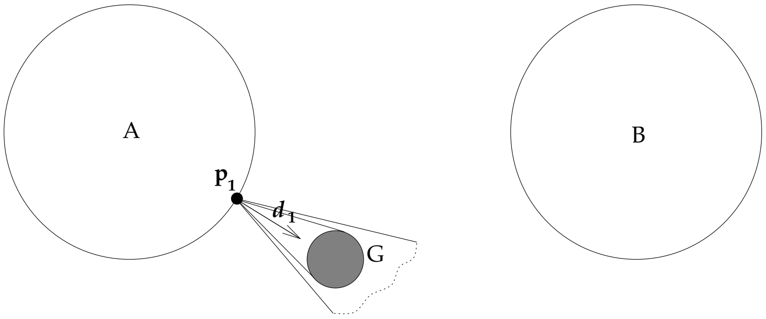
Given a motion command and an extension set, it is easy to compute a set $\mathcal{F}$ of landmarks consisting only of unreachable or failure landmarks. $\mathcal{F}$ is constructed by finding those landmarks that do not intersect with the weak preimage of the extension set for this motion command. We cannot tell whether a landmark in $\mathcal{F}$ belongs to the unreachable or the failure set, but we know that the robot cannot enter any of the unreachable landmarks. Therefore, if the robot during its motion enters a landmark in $\mathcal{F}$, this has to be a failure landmark, so the robot should stop and declare failure.

Selecting the commanded velocity direction Choosing d in such a way that the robot will either hit a landmark in the extension set or a landmark in the failure set is a much harder problem. This direction may not be the one that maximizes the chances of the robot to hit the extension set, but it may be preferable to one that could allow the robot to fail in some inappropriate way. As long as the robot has some positive probability to hit the extension set, we may be better off to choose strategies that offer recoverable failure.

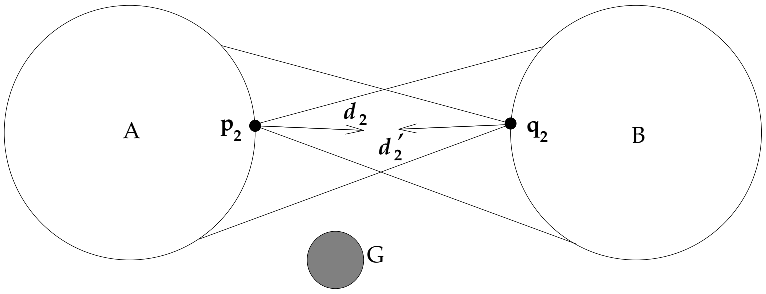
Consider the example of Figure 6.15. The workspace consists of two big landmark disks A and B and a small landmark disk G that is also the goal of the robot. The robot begins its motion from some point within the landmark disk A. Let us also assume that the directional uncertainty is big enough, so that no preimage of G can intersect either A or B, and, consequently, no guaranteed plan exists. If we let the greedy algorithm presented above create a maximum-likelihood-of-success non-guaranteed plan, we get the plan shown in Figure 6.15(a). The robot is asked to move initially to point $\mathbf{p}_1$, and then follow the commanded direction d_1. In the figure we also show the forward projection of $\mathbf{p}_1$ for the given motion command, from which it is evident that there is some chance that the robot may move past G into terra incognita.

In Figure 6.15(b) we show an alternative plan: whenever the robot is in A it moves to point $\mathbf{p}_2$ and follows d_2; whenever it is in B, it moves to $\mathbf{q}_2$ and follows d'_2. Both forward projections of $\mathbf{p}_2$ for d_2 and $\mathbf{q}_2$ for d'_2 are bounded, the first by B, the second by A. Thus, the robot will bounce between A and B without possibility to get lost. Unfortunately, the goal landmark G lies outside both forward projections, so the robot will never reach it.

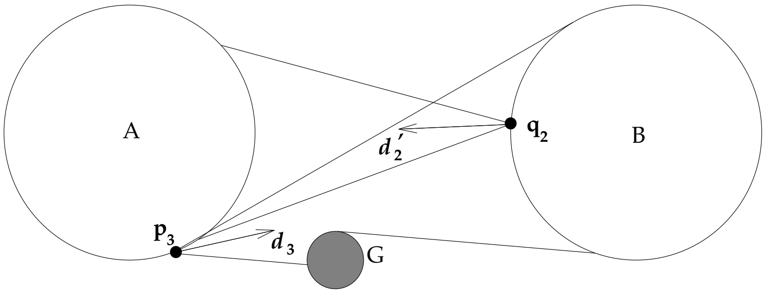
By selecting a different exit point $\mathbf{p}_3$ and commanded direction d_3 for disk A, it is possible to make G part of the forward projection of A's exit command (Figure 6.15(c)). Now there exists a positive probability that the robot will achieve G during its bouncing between A and B. The number of execution steps before the robot gets to G is not bounded, but the success probability of this plan tends to one as the number of execution steps becomes arbitrarily large. The expected number of execution steps is bounded. Such plans are called *probabilistically guaranteed* and have been investigated by Erdmann [34].



(a) Possible unrecognizable failure



(b) Infinite loop with no probability of success



(c) Loop with positive probability of success

Figure 6.15: Non-guaranteed plans

A graph representation of EDR plans Although in the above simple example it is easy to find a probabilistically guaranteed plan, in general it is quite hard. If we employ a backward chaining technique, in the worst case we will have to test the union of the extension set with all possible combinations of intermediate-goal disks. The problem is simplified if we assume the generalized landmarks model of Subsection 5.5.1. In this case the robot does not have perfect navigation within landmark disks; it can only move reliably into a specific region within each landmark disk. Without the problem of exit region selection within a landmark disk, a forward chaining algorithm is more appropriate.

Given an initial region (or an exit region if the robot is already within a landmark) an algorithm very similar to the one presented in Chapter 4 can be used to compute in polynomial time the *omnidirectional forward projection* of the region. The algorithm returns the list of all distinct landmark sets which completely block the forward projection (if any), each associated with a certain interval of commanded directions of motion. If the robot follows one of these commanded velocity directions, it is guaranteed to terminate its motion into one of the landmark disks of the associated termination set. Let us call this list the *blocking list* $\mathcal{BL}(L)$, where L is some landmark disk. We can precompute $\mathcal{BL}(L)$ once for each landmark in the workspace in $O(\ell^5 \log \ell)$ total time.

Then, EDR planning is performed by choosing one element from each blocking list, which is equivalent to prescribing an exit motion command for each landmark. Instead of a specific commanded direction of motion, these commands determine an interval of possible directions. Each such direction is guaranteed to make the robot terminate its motion into another landmark region, if the motion starts from within the exit region of the landmark. If the blocking list of a landmark is empty, then there is no safe motion command from this landmark, thus no motion command can be prescribed. If such a landmark is not a goal landmark, it is called a *hole*, because it causes premature termination of the execution of a plan. A probabilistically guaranteed plan must prevent the robot from reaching such landmarks.

An EDR plan is a collection of landmark sets consisting of one element from each blocking list. Every possible EDR plan can be constructed in this way, therefore

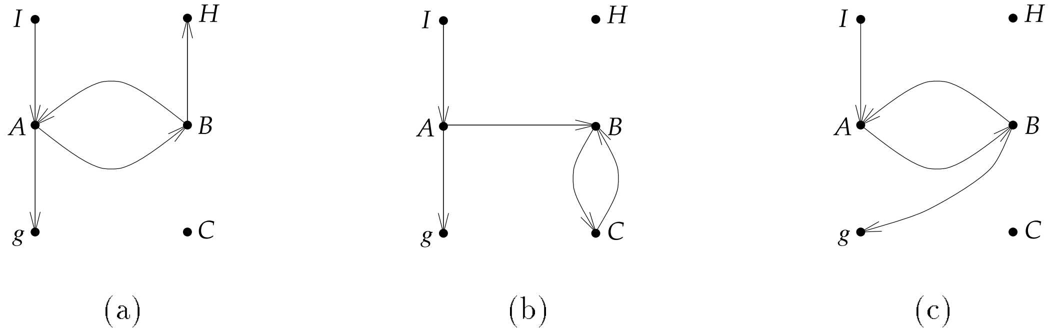


Figure 6.16: Graph representation of EDR plans

the blocking lists are a complete representation of all possible EDR plans in the workspace. The number of elements in each blocking list is a polynomial function of the complexity of the workspace, therefore the blocking lists representation not only is complete, but polynomial too. The total number of EDR plans, though, is an exponential function of the workspace complexity.

An EDR plan is probabilistically guaranteed, if during its execution the probability to hit a hole is zero, and the probability to hit some goal landmark is positive. We can use a directed graph to represent an EDR plan. Each node represents a landmark. Arcs represent possible transitions from landmark to landmark. The nodes representing the goal landmarks are called *goal nodes*. Non-goal nodes with no arcs coming out of them represent the hole landmarks and are called *hole nodes*. One or more nodes represent the possible initial state of the robot and are called *initial nodes*. Figure 6.16 presents three EDR plans. Plan (a) is not guaranteed because there exists a path from the initial node $\mathcal{I}$ to the hole node H. Plan (b) is not guaranteed because a robot starting from $\mathcal{I}$ may get into the cycle BC with no possibility to escape. Plan (c) corresponds to the example of Figure 6.15 and is probabilistically guaranteed; the robot will eventually reach $\mathcal{G}$ after an unbounded number of AB cycles.

Theorem 6.6 *An EDR plan is probabilistically guaranteed, if and only if all the maximal connected subgraphs that contain the initial nodes contain at least one goal node but no hole nodes. This can be decided in linear time.*

An alternative formulation of the above theorem uses Markov chains. Indeed, every directed graph is equivalent to a Markov chain with the arcs being the transition probabilities. In this case, we say that an EDR plan is probabilistically guaranteed, if and only if all recurrent classes that contain the initial nodes also contain a goal node, but no hole node.

Various heuristics can be used in an A^*-search of the exponential space of all EDR plans for probabilistically guaranteed ones. However, it is unlikely that polynomial search algorithms do exist.

The set of guaranteed plans with a bounded number of execution steps is a subset of the set of probabilistically guaranteed plans.

Theorem 6.7 *An EDR plan is guaranteed and has a bounded number of execution steps, if and only if all the maximal connected subgraphs that contain the initial nodes contain at least one goal node, no hole nodes, and no cycles.*

We have already shown that such plans can be found in polynomial time.

6.4 Other applications of multiparametric planning

The planning methods described above can be used to solve a multitude of additional practical planning problems. In this section we describe how this can be done in several interesting cases.

6.4.1 Planning with anisotropic uncertainty

In Figure 6.17 we show the $d\theta$-space and the event curves. If the uncertainty of the robot is fixed at a nominal value of θ_f, the intersections of these curves with the line $\theta = \theta_f$ correspond to the critical events of the monoparametric case described in Chapter 4. In some applications, however, the directional uncertainty depends on the commanded direction of motion. For example, this can happen when a wheeled robot moves on a carpet, when an underwater robot navigates in the presence of currents, or more generally in any flow field. If we know the function $\theta = f(d)$, then we can

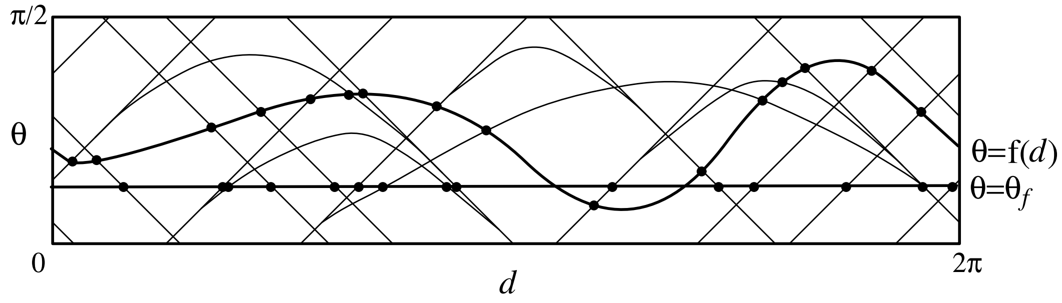


Figure 6.17: Modeling anisotropic directional uncertainty

compute its intersections with the event curves. These intersections correspond to critical events that partition the curve $\theta = f(d)$ in regular subsets. We need to compute only one preimage per subset.

6.4.2 Unexpected obstacle avoidance

Assume that the robot workspace may contain unexpected obstacles, i.e., obstacle regions that are not in the planner's model. Assume further that the robot can detect the unexpected obstacles interfering with a motion command just before this command is executed.

Figure 6.18 shows an environment with an extension disk in white, an intermediate-goal disk in grey, and an unexpected obstacle in black. If θ_f is the given constant directional uncertainty of the robot, the planner of Chapter 4 computes a single preimage $\mathcal{P}_f$ of the extension disk that intersects the intermediate-goal disk, and provides the corresponding value of d (e.g., the direction d_f shown in the figure). However, if the robot executes a motion commanded along d_f starting from any point in the intersection of $\mathcal{P}_f$ with the intermediate-goal disk, it may hit the unexpected obstacle.

Instead, we can compute the maximal value of θ, θ_m, for which there exists a preimage $\mathcal{P}_m$ of the goal extension that intersects the intermediate-goal disk, yielding a direction of motion d_m. Let $C(d_m, \theta_m)$ be the cone of half-angle θ_m about d_m. Let the robot move in the perfect-control mode to the intersection of $\mathcal{P}_m$ with the

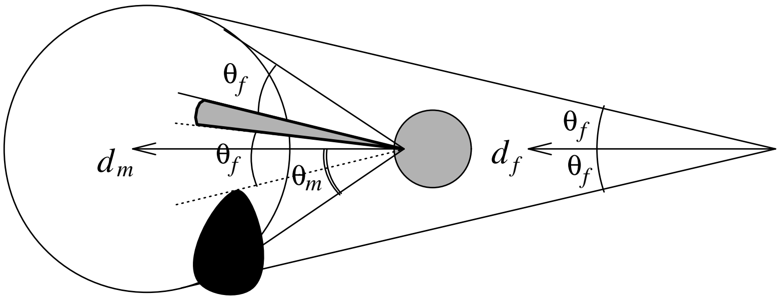


Figure 6.18: Avoiding unexpected obstacles

intermediate-goal disk before executing an imperfect-control motion command of uncertainty θ_f toward the goal extension. In the absence of unexpected obstacles, the axis of any cone of half-angle θ_f contained in $C(d_m, \theta_m)$ is a commanded direction of motion guaranteed to achieve the goal extension with directional uncertainty θ_f. In the presence of unexpected obstacles, as is the case in Figure 6.18, if there exists a cone of half-angle θ_f included in $C(d_m, \theta_m)$ but not intersecting the unexpected obstacles, its axis is a valid commanded direction. The grey cone in Figure 6.18 contains all valid commanded directions for this example.

The major benefit of this planning technique is that it avoids unnecessary replanning if something unexpected occurs. Instead of committing to specific imperfect-mode commands, the robot computes a set of guaranteed commands for each landmark disk, and makes a choice only when necessary. Thus, if some of the commands have become non-guaranteed due to an unexpected event, the robot can exclude them from its available choices.

6.4.3 Adjustable sensor sensitivity

If the sensor that detects the landmarks has adjustable sensitivity, then effectively the radius of the landmark disks is adjustable. The events that produce the critical curves remain the same, but the critical curves depict now the relation between sensor sensitivity κ and the commanded direction of motion d. Consider an I-Include critical curve for example. Let $\eta_i = \mathcal{H}_i(\kappa)$ denote the dependence of the radius of an extension set landmark disk D_i on the sensor sensitivity κ. Call the radius of the initial-region

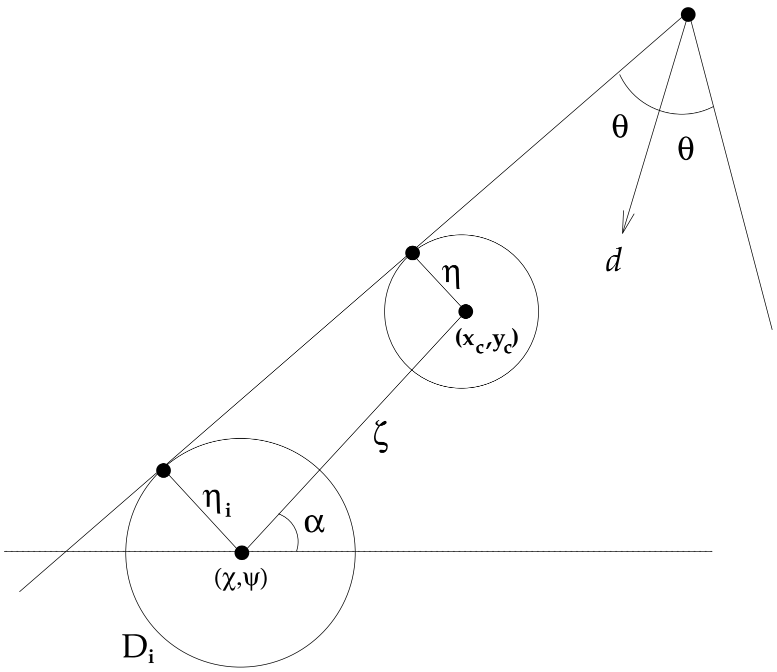


Figure 6.19: Deriving the critical curve of an *Include* event

disk to be covered η, the distance of the two disks ζ, and the slope of the line that connects their centers α (see Figure 6.19). The slope of the left ray is equal to $\alpha - \arcsin((\mathcal{H}_i(\kappa) - \eta)/\zeta)$. It is also equal to $d + \pi - \theta$. Thus, the I-Include critical curve has the following equation: $\kappa = \mathcal{H}_i^{-1}(\eta - \zeta \sin(\alpha + \theta - d))$. Other critical curves have analogous equations, and one can proceed to produce minimum cost plans, using the algorithms described in this paper.

6.4.4 Robot cooperation

Another parameter that can affect the preimage is the location of the landmarks, which determines the location of the landmark disks in the workspace. Consider the following simple case: Along with fixed landmarks, the workspace also possesses a movable landmark, that can position itself with infinite precision anywhere in a specified subset of the free space. Motion of this landmark can be effected with the help of a second highly precise robot. For example, if we constrain the motion of this robot within a particular landmark region, it will be perfectly precise as per

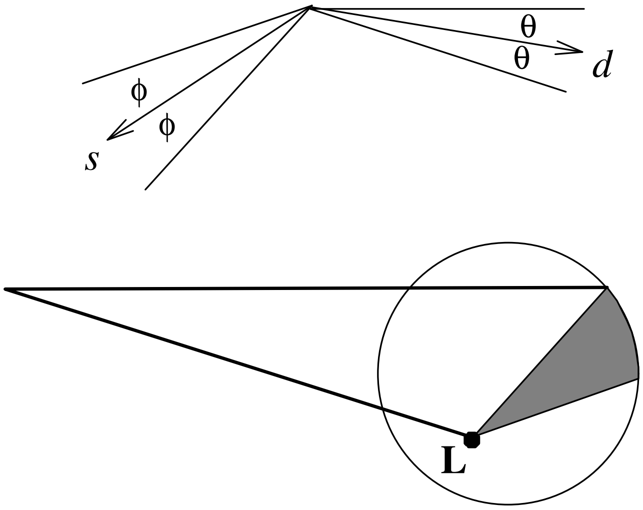


Figure 6.20: Limited sensing angle and the active region

our assumptions. In fact this second robot itself may play the role of the movable landmark, in order to help the initial robot navigate safely. If we denote the position of the movable landmark with (χ, ψ), then critical events correspond to surfaces in the χ-ψ-d space. All critical events that do not involve the movable landmark create critical planes parallel to the χ-ψ plane, since they do not involve either χ or ψ. Critical events that involve the movable landmark, however, create complicated surfaces. For example, referring again to Figure 6.19, the critical surface that corresponds to the I-Include event of the figure, is the solution to the equation $\zeta(\chi, \psi)\sin(\alpha(\chi, \psi)+\theta-d) = \eta - \eta_i$, with $\zeta(\chi, \psi) = ((\chi - x_c)^2 + (\psi - y_c)^2)^{1/2}$ and $\alpha(\chi, \psi) = \arctan(y_c - \psi, x_c - \chi)$, where x_c and y_c are the coordinates of the center of the disk to be covered, η its radius, and η_i the radius of the movable landmark.

6.4.5 Directional sensing

Up to now we have assumed that the sensor of the robot is omni-directional. Nevertheless, it is more realistic to restrict sensing within a cone of half-angle ϕ, about the *aiming direction* s of the sensor. Let us call this cone the *visibility cone*.

Given ϕ, the choice of s determines which part of a landmark region can serve as termination condition for a motion, during which the robot aims its sensor along s.[8] We call this part the *active part* of the landmark region. For example, look at Figure 6.20. Here we show a circular landmark region with a point landmark (marked **L** in the figure) located in its interior. Its active part (the dark shaded region in the figure) for some choice of s is its intersection with an inverted visibility cone, anchored at the landmark and with its axis along $s + \pi$. This computation is valid for arbitrary landmark region shapes and landmark locations (inside or outside the landmark region). For every choice of s, instead of finding the preimage of the entire landmark regions in the extension set, we compute the preimage of their active parts. These are generalized polygons and can be handled by the planner as claimed in Subsection 5.1. However, the relations between d and s that determine the critical curves are significantly more complicated. In Figure 6.20 we have outlined the preimage of the active region for the shown choices of d and s.

A motion command is a triplet $(d, s, \mathcal{L})$. As with d, usually there is no cost associated with the choice of s.[9] Thus, we need not sweep the ds-plane in a particular direction; we just need to visit each cell once. If a cell is found for which the preimage covers all initial-region disks, we are done. Otherwise, we need not stop visiting cells as soon as we find a cell for which the preimage intersects an intermediate-goal disk. Instead, we mark the intersected disk, and we associate with it the cell for which the disk was intersected. This information will be used to create the motion command associated with this disk. If all cells are visited and no preimage includes all initial-region disks, all intersected intermediate goal disks become part of the new extension set. If we mark visited cells so that we visit them only once over all iterations of the algorithm, we can produce multi-step plans with the minimum number of steps in $O(\ell^6 \log \ell)$ time.

---

[8] We assume that s remains constant during the execution of a single motion command.

[9] Of course one could argue that the robot should prefer to aim its sensor in the direction of its motion to avoid hitting unexpected obstacles, but we can assume that there are other sensors dedicated to this task.

6.4.6 Motions in three-dimensional configuration spaces

Consider a three-dimensional configuration space containing a point robot. Obstacle and landmark regions are three-dimensional objects. Imperfect-control motions are assumed to be always planar, in the sense that control errors cannot make the robot move outside the chosen plane of motion. Within this plane, though, the standard planar control uncertainty model applies. For example, the above model can describe compliant motions on planar obstacle surfaces. On the other hand, perfect-control motions can be fully three-dimensional. We further assume that we can compute the intersections of the valid planes of motion of the robot with all obstacle, landmark, goal, and initial regions. The available planes of motion are represented by the normal vector $\vec{p}$.

Under these assumptions, an imperfect-control motion command is determined by the triplet $(\vec{p}, d, \mathcal{L})$ assuming constant directional uncertainty θ. The algorithms of this chapter are directly applicable, if the critical surfaces (two dimensional surfaces in the three-dimensional $\vec{p}d$-space) can be computed.

6.5 Conclusion

In this chapter we presented algorithms for the efficient computation and representation of preimages that depend on more than one parameters. We examined in detail the case of planning with controllable directional uncertainty and discussed several frameworks where such planning may be useful. Similar algorithms can be developed for the dependence of the preimage on any number of parameters, however, their computational complexity is an exponential function of the number of parameters considered. Furthermore, computing critical hyperplanes becomes more complicated as the dimension of the search space grows. Nevertheless, using numerical techniques to compute critical hyperplanes and their intersections, it is possible to develop efficient planners for a variety of interesting planning problems.

Chapter 7

Experiments with a Real Robot

In Chapters 4, 5, and 6 we argue that the only "practical" way to deal with the problem of planning with uncertainty is to adopt simplified formulations of the problem which can be solved with polynomial algorithms. We have presented several variants of such simplified formulations, along with correct, complete, and polynomial planning algorithms. In this chapter we validate our claim of "practicality" by using these planning techniques in experiments with a real mobile robot.

To make the vague notion of practicality more concrete, we identify four major components of it:

- **Cost:** Our planners are based on a specific set of assumptions regarding the robot and the workspace. We have to make sure that these assumptions are valid, by engineering, if necessary, the robot and the workspace. For the planners to be practical, the cost of such engineering must be low.

- **Completeness:** If the assumptions are valid, the planners are complete. However, to make the problem tractable, we use only a fraction of the information that is available to the robot. Therefore, our planners can solve fewer problems than more sophisticated (and slower) planners can. The set of problems solvable by them must cover the majority of the tasks that the robot may be required to perform.

- **Correctness:** Theoretically, no plan should fail. Nevertheless, it is not always possible to completely satisfy all the assumptions. For example, it is not physically possible to create regions with zero position uncertainty. The percentage of successfully executed plans measures our ability to satisfy the necessary assumptions at a given cost.

- **Robustness:** When failures occur, a practical system should be able to handle them gracefully. In the experiments, we investigate to what extent our system is able to do so.

We conducted two sets of experiments with two different landmark designs. In the first set, the landmarks are triangular prisms with one missing face, standing vertically on the floor. The location and orientation of the prisms is precisely known. The robot emits a horizontal plane of laser light, whose reflection on the landmarks is captured by a camera on the robot. By analyzing the reflection, the robot can detect and recognize a landmark, and localize itself with respect to the landmark. Since the landmark coordinates are accurately known, the robot can also localize itself in the workspace frame of reference. For this set of experiments we make use of the standard landmark definition that requires perfect localization precision in the landmark regions.

In the second set of experiments, the landmarks are marks on the ceiling. Images from a camera (mounted vertically on the robot and aiming straight up) are used for detection, recognition, and localization. Here, we use the notion of generalized landmarks (see 5.5.1); the only requirement regarding localization is that for each landmark region there exist a subregion which can be reliably achieved, once the landmark has been detected and recognized.

7.1 The robot

The robot we use is NOMAD-200 from Nomadic Technologies. NOMAD is a three-wheeled, cylindrical mobile robot. It is relatively small, with a radius of about 1 foot and a height of about 2.5 feet. Barring slipping or jumping, the absolute orientation of its body remains constant. The three wheels can rotate to any orientation relative

Figure 7.1: NOMAD-200

to the body, but they always remain aligned with each other. The robot translates along the direction of the wheels. A plate holding sensory equipment, called the *turret*, is mounted on top of the body and can rotate freely.

Since NOMAD is a wheeled robot, its motions are subject to non-holonomic constraints. However, its wheels can assume any orientation, hence, the robot has zero turning radius. Thus, in the absence of obstacles it can move between any two points on the floor along a straight line. Its configuration at each instant is given by a quadruplet of coordinates (x, y, δ, α), that define its x- and y-position, the absolute orientation of the wheels, and the absolute orientation of the turret respectively.

NOMAD is equipped with many sensors (odometric sensors, infrared and sonar proximity sensors, touch-detecting bumpers, a laser with a cylindrical lens that can emit a plane of monochromatic light, and two cameras that can be attached to the turret in a variety of orientations). Of all these sources of information, we must select a subset that satisfies the assumptions required by our planners.

7.2 Landmark design

During the landmark design phase, we have to take into account three sources of constraints: (a) the model of the robot and the workspace used by the planning algorithms, (b) the status of the implemented algorithms, and (c) the availability of hardware and software for real-time landmark recognition.

We conduct the experiments in an indoor office-type environment. We obtain very accurate workspace modeling through precise prior measurements. Small errors during these measurements are taken into account by slightly growing the obstacles in the model used by the planner.

The landmarks are physical features of the robot workspace. Each landmark induces a region in the workspace (the landmark region), such that a robot standing at any point of the region is able to (a) detect the landmark, (b) recognize the landmark, and (c) localize itself with respect to the workspace. All three operations must be extremely reliable. In addition, detection must be fast (ideally instantaneous) and localization precise. These properties lead us to design artificial landmarks rather than use "natural" ones (e.g., corners between walls).

An important consideration is the source of orientation information for the robot. The planner assumes that the robot is aware of its orientation in the global frame of reference, because the commanded direction of motion is specified in this frame. NOMAD's actual direction of motion is determined by the absolute orientation of the wheels δ. A compass, attached to the wheels, would be the straightforward way to get an estimate of δ, but in an office environment it would be subject to big errors, caused by the electromagnetic fields of the various office equipment. For this reason, we get orientation information from the landmarks during the localization phase.

Even though our algorithms apply readily to overlapping generalized polygonal objects, the implemented version of them requires that landmark regions and C-obstacles are circular, and that landmark regions and C-obstacles do not overlap.

7.3 Bounding the directional uncertainty

The orientation reading we get from the localization process, is an estimate $\hat{\alpha}$ of the absolute orientation of the turret. (The sensing equipment is mounted on the turret.) The encoders on the motors that turn the turret and the wheels provide estimates for the relative angles between the turret and the body (α_{od}) and the wheels and the body (δ_{od}). Thus, our estimate of the global orientation of the wheels (which determines the actual direction of motion) is computed as $\hat{\delta} = \hat{\alpha} - \alpha_{od} + \delta_{od}$. The error $\epsilon_{\delta} = |\hat{\delta} - \delta|$ is the *aiming uncertainty* of the system, and it is the sum of three independent errors, the localization error, the error of the wheel encoder, and the error of the turret encoder. When the robot moves, it cannot follow a straight line because of *control uncertainty*. Possible causes are slips, jumps, uneven floor, or wheels that cannot be kept absolutely steady relative to the body. The combination of aiming and control uncertainty constitutes the *directional uncertainty* of the system. To produce guaranteed plans, the directional uncertainty must be bounded by the value θ used by the planner.

Assuming no slips in the motors that turn the wheels and the turret, the values of α_{od} and δ_{od} are accurate up to the resolution of the encoders. The error they contribute is negligible with respect to the other sources of directional error. The experimental bound of the orientation error of the localization algorithm is $\pm 3^\circ$. Regarding the control uncertainty of NOMAD we found that it is bounded by 1° when the robot moves on the floor of the laboratory. This control uncertainty is basically caused by the uneven floor (linoleum and carpet with metallic joints between them).

There is one additional source of directional uncertainty: it is the error in the initial orientation of the robot, if the initial location does not lie within a landmark region. This error, of course, is eliminated once the robot localizes itself once. In our experiments, the robot is transported manually to its initial position and orientation.

In the first set of experiments, we make sure that the initial orientation has an error of less than $\pm 5°$, so it supersedes the aiming error. The value of θ used is $6°$. In the second set of experiments, we make sure that the initial directional uncertainty of the robot lies within the localization algorithm uncertainty bounds, so we may use $\theta = 4°$.

7.4 Experiments with prismatic landmarks

The simplest landmark design that can provide position and orientation information in two dimensions is a line and a point with known coordinates in the global frame of reference. A robot that can sense its orientation relative to the line and its distance from the point can compute its own global coordinates by means of a simple transformation. In our case, we also require that the landmarks possess a unique identifying characteristic. A design that satisfies all these requirements is the following.

The landmarks are constructed as *vertical corners*, i.e., two intersecting vertical planes (*facets*) of known orientations. The laser mounted on the turret of NOMAD emits a horizontal plane of light at a height of 2.5 feet above the floor. A camera, mounted on the turret and tilted downwards, detects the intersection of this plane with the facets of each landmark. This intersection consists of two line segments with a common endpoint (see Figure 7.2). We can estimate the orientation of the robot relative to each segment and the distance of the robot from the intersection of the segments (the *vertex*) very accurately. The difference of the two relative orientations is the angle between the facets, and can be used for landmark recognition. After a landmark is recognized, we can read from a landmark database the absolute orientations of the two facets and the absolute location of the vertex. From these data we can deduce the absolute position and orientation of the robot.

We create landmarks by putting together white $8in \times 40in$ rectangular foam boards to form an angle that stands vertically on the floor. Since no two landmarks should be confusable, each landmark must have a distinct angle (the *characteristic angle*). For this reason we cannot use wall corners as landmarks. Furthermore, the characteristic angles we use, must not be observable by the laser-camera system when no landmarks

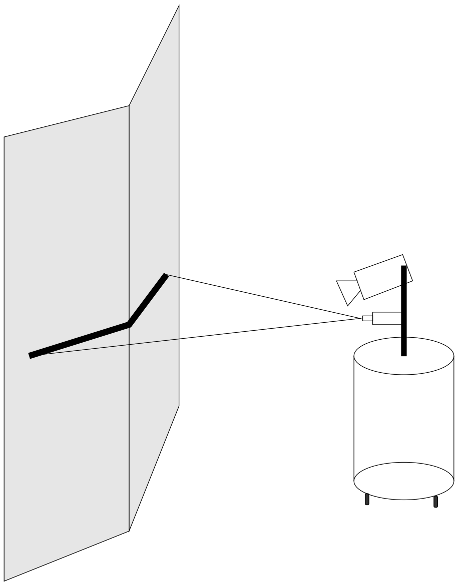


Figure 7.2: A prismatic landmark

are present. Wall corners are detectable by the laser sensor, so our landmarks must avoid using characteristic angles that are close to 90°.

7.4.1 Computing the landmark region

Although the camera is pointed downwards, its angle of vision is such that it can theoretically see the laser plane out to infinity. However, reliable recognition requires that the images of the two landmark segments on the view plane of the camera have a minimum length. This constraint can be used to derive a theoretical outer bound of the landmark region. Moreover, other practical constraints, like lens aperture and focus, illumination, and reflectivity of the surfaces, cause the quality of the data to vary with the location of the robot. The work of Takeda on *sensory uncertainty fields* (SUF's) [91] can be used to select regions where localization uncertainty does not exceed a certain bound. However, since our planner currently handles only circular landmark regions, we would still have to extract a circular region from the sensory uncertainty field. Instead, we use a simpler experimental approach. We determine a minimum and a maximum distance between the vertex of the landmark and the center of the robot (30 and 50 inches respectively), and a minimum angle between the

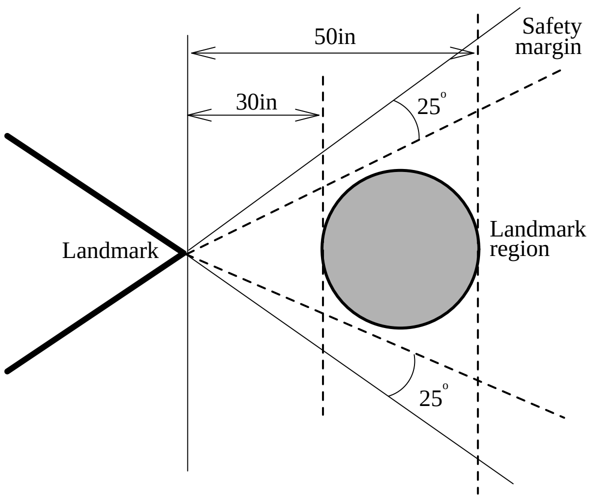


Figure 7.3: Deriving the landmark region

line that connects the center of the robot with the vertex and each of the facets (25°), for which the accuracy of the data is acceptable. We define the landmark region as the biggest disk all the points of which satisfy the above constraints (see Figure 7.3).

7.4.2 Recognition and localization

An intensity filter can easily isolate the pixels that correspond to the reflections of the laser light on objects. The camera is calibrated, so that each pixel in the view frame-buffer can be transformed into the polar coordinates of the workspace point that corresponds to that pixel (in the frame of reference of the robot). Essentially, we get a sample of points from the line segments that form the intersection of the laser plane with the objects in the field of view of the camera. The resolution of the sample (i.e., how many pixels correspond to a sample point) can be defined by the user. High resolution results in better accuracy, but makes the recognition algorithm slow. We found that 60 points spanning the 30° of the camera angle of view give us satisfactory accuracy (for viewing from within landmark regions) and a fast enough recognition algorithm.

The next step is called *segmentation* and allows us to connect the sample points into line segments. We use a Kalman filtering algorithm that allows us to get the segments that fit our sample points best. Adjacent segments are joined in an *elbow* and are returned as potential landmarks. The information contained in an elbow structure consists of the orientations of the two segments and the coordinates of the intersection of the segments (all in the frame of reference of the camera). From the two orientations we extract the angle of the elbow, and we match it against the characteristic angles of the landmarks. If the difference is smaller than the accuracy of the algorithm, we have a match. In practice, we achieve good accuracy (less than 10°) in computing the angle of elbows from points within the landmark region. This allows us to use several distinguishable landmarks of 60°, 90°, 120°, 240°, 270°, and 300° (their angles should be at least 20° apart).

After a matching landmark has been found, we enter the localization phase. We select the configuration of the landmark (position and orientation) in the frame of reference of the camera, so that it best fits the sample data. Since we already know the configuration of the landmark in the workspace frame, we can get the configuration of the camera in the workspace frame of reference with a simple transformation. The algorithm also returns error bounds from the fitting process, which are 2.5 inches for the position and ±3° for the orientation. The value of θ used is 6°.

7.4.3 Deviations from the planning model

The prismatic landmark design does not have all the properties regarding landmarks and landmark regions required by the planning model. In this section we show how we can work around these deviations by augmenting the behavior of the robot at execution time.

Imprecise localization The planning model we use in this set of experiments requires infinitely precise localization within a landmark region. The error bound of the localization algorithm was experimentally found to be 2.5 inches. That is, the actual position of the robot lies in a disk with a 2.5-inch radius centered at the measured robot position. For a motion command to be actually guaranteed, all possible initial robot positions must lie within the exit region associated with this command. If a disk with a 2.5-inch radius cannot be inscribed in the exit region, then the motion command (hence, the generated plan) may fail. Our planner selects among all possible exit regions the one with the largest area, but it does not discard exit regions that are too narrow. Therefore, whether a plan is guaranteed or not is checked only after the creation of the plan. It may be possible to guarantee at planning time that exit regions will be large enough to contain the error disk (by adding additional critical events that correspond to this property of exit regions), but we have not investigated this possibility. In the second set of experiments we avoid this source of errors by using generalized landmarks.

In this set of experiments, we make sure that all exit regions are large enough before accepting a plan. At execution time, whenever NOMAD detects and recognizes a landmark, it localizes itself and attempts to move to the center of the maximum circle that can be inscribed in the exit region. It repeats the same motion, until all possible actual positions of the robot lie within the exit region, or until some prespecified number of attempts has been made.

Dependency of landmark detection on robot orientation The construction of the landmark regions assumes that the camera is pointed towards the vertex of a landmark. Although one of the extensions of the planner (see Section 6.4.5) solves specifically this problem, it has not been implemented yet. Therefore, we have to find another way for the robot to look towards a direction which guarantees that one of the landmarks in the termination condition will become visible.

Let us assume for the moment that there is only one landmark in the termination condition. The forward projection of the current position of the robot for the chosen direction of motion must be completely blocked by the landmark disk, which means that a left and a right ray of the forward projection cut through the landmark disk (see Figure 7.4). The robot is guaranteed to cross any line segment that connects the left and the right ray with its endpoints in the landmark disk. Consider one such segment, such that the straight line it defines passes through the the vertex of the landmark. If the camera is pointed along this line, the robot is guaranteed to see the landmark while it is still in the landmark region. Since the view angle of the camera

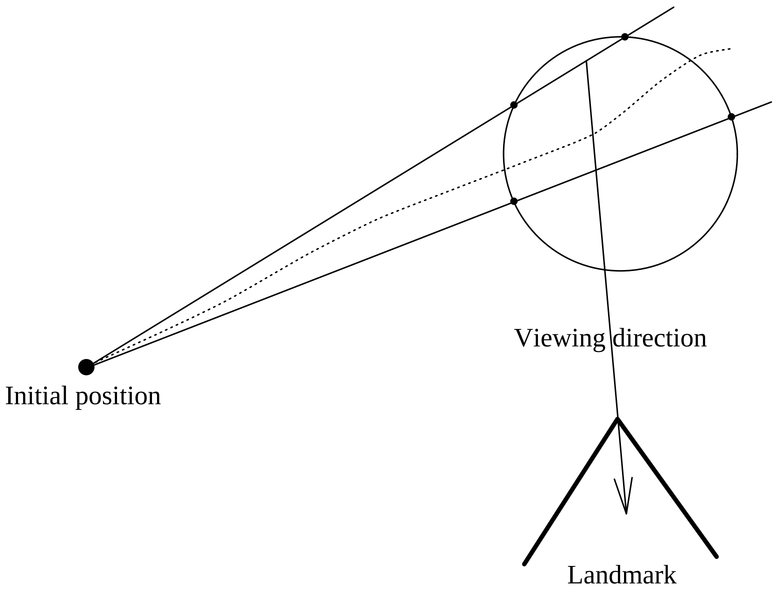


Figure 7.4: The guaranteed viewing direction

(30°) is larger than the angle of the directional uncertainty cone (12°), the landmark is detected along this line.

If the termination condition contains more than one landmark, the robot has to select one of them in order to aim the camera towards it. For this decision we make use of the odometric sensor. First, we eliminate the landmarks in the termination condition that do not intersect with the forward projection, because they cannot be reached. Then, we sort the remaining landmarks along the commanded direction of motion of the robot. At the beginning, the camera is pointed towards the first landmark. When it is determined that the robot is past that landmark region (by use of the odometric sensor), the camera is turned towards the next one, and so on. If the landmark regions are separable along the commanded direction of motion (i.e., only one is detectable at a time), then the plan is guaranteed.

Our planner cannot ensure that the plans it creates have the above property. However, after we have a plan, it is easy to check whether the above condition is satisfied. While this is a limitation of our experiment, it is easy to arrange landmarks in the workspace so that the condition is met. The ceiling-landmark design avoids this problem altogether.

Non-instantaneous detection Another deviation from the assumptions of the planner is that the detection algorithm does not run instantaneously. Its execution requires about 0.2 seconds. To guarantee detection, the landmark must remain within the camera range for at least 0.2 seconds. While it is certainly possible to work out whether this will happen or not, we do not present such calculations here, because the ceiling-landmark design considerably simplifies this problem.

Non-instantaneous stopping Once the robot detects a landmark, it must stop and localize itself. Since deceleration is not infinite, it is possible that the robot will not be able to view the landmark after it has completely stopped. If this occurs, the robot scans the workspace (by turning its turret and running the detection algorithm) until it sees the landmark again. If this approach fails, the robot backtracks a little (one inch) and repeats the scanning. If this fails again, the robot declares failure.

7.4.4 Plan execution

We begin by building a model out of the physical arrangement of the workspace. We describe above how we create the landmark regions. Since the obstacles in our model need to be circular, we cover all physical obstacles[1] with an arrangement of circular disks. In our planning model we have to compute C-obstacles. For a circular robot in two dimensions this is easy to do: We just grow the obstacles by the radius of the robot.

The planner is called and returns a plan (or failure). The plan is preprocessed, to associate one or more viewing directions with each command, and then it is executed:

1. The wheels are turned towards the initial commanded direction, and the turret is turned towards the first viewing direction.
2. The robot starts moving forward, continuously checking for the presence of landmarks. If the targeted landmark is found, motion stops immediately. If a maximum distance is traveled without detecting the landmark, the turret is

---

[1] The foam boards that provide the landmarks are obstacles, too.

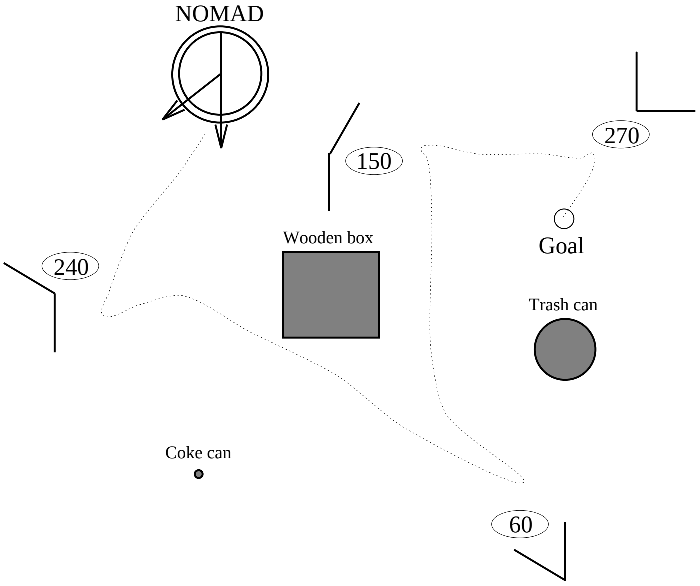


Figure 7.5: The layout of the workspace

turned to face the next landmark. If no more landmarks exist in the termination condition, the robot considers itself lost.

3. If a termination condition landmark is detected, the turret is turned towards its vertex (for better viewing). Localization gives estimates for the current robot position and orientation and error bounds.

4. If the error disk (the locus of all possible robot locations) is contained in the exit region of the landmark, then the robot proceeds to execute the command associated with the landmark. Otherwise, the robot moves towards the center of the exit region, always facing the vertex of the landmark; after it gets there, it localizes itself once again. If the error disk is still not contained in the exit region, the robot keeps trying to get to the center of the region and relocalize. It stops, when the exit region is achieved with certainty, or when a maximum number of attempts has been performed.

5. If the exit region is in the goal, the execution terminates successfully.

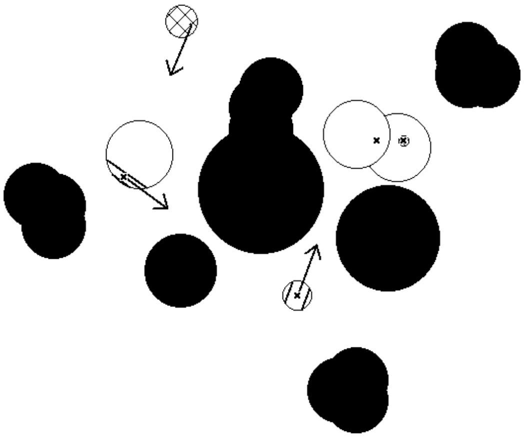


Figure 7.6: The configuration-space model used by the planning algorithm

As an example, consider the workspace in Figure 7.5. There are four landmarks, and three obstacles (a box, a trash can, and a Coke can). The initial position has a 5-inch uncertainty bound, and the goal is a 2-inch disk located right above the trash can. The number beside each landmark is the characteristic angle. Figure 7.6 shows the model input to the planner and the constructed three-step plan generated by the planner.

NOMAD was successful in getting to and stopping in the goal 22 out of the 25 times we ran the experiment. First it moves towards *landmark-240*, then it passes between the Coke can and the box, relocalizes itself at *landmark-60*, and heads towards *landmark-150.* After relocalization at this landmark, the turret is turned towards *landmark-270*, which helps the robot achieve the goal within the localization algorithm accuracy. Twice, NOMAD failed to enter the exit region of *landmark-240* because of its small width. After five attempts, we instructed it to continue with the execution of the plan, and as a result it slightly touched the wooden box during its motion. Once, NOMAD ran past *landmark-240* without detecting it.

7.5 Experiments with ceiling landmarks

The major source of complications in our first set of experiments is the dependence of landmark detection on the orientation of the camera. Furthermore, the landmarks we insert in the workspace are rather bulky and reduce the ability of the robot to move in its workspace. Our second landmark design completely avoids both problems.

Landmarks are created by fixing on the ceiling black-and-white patterns (see Figure 7.7) at well-defined locations and orientations in the workspace coordinate system. These patterns are sensed by an upward-looking camera fixed on the robot with its axis collinear to the robot's axis of rotation. All landmark patterns are designed to have the same size in the camera image. A landmark pattern consists of three elements: (1) an outer thick circle (black), (2) an opening into this circle (white), and (3) an inner 3×3 grid of black and white tiles aligned with the opening in the circle. The first two elements are the same for all landmarks. The third is unique to every landmark (thus, $2^9 = 512$ distinct landmarks are possible).

Another problem we have in the experiments with the prismatic landmarks is the requirement for infinitely precise localization. During this set of experiments we use the notion of generalized landmarks. Each landmark must be detected and recognized whenever the robot enters its landmark region. Additionally, there should exist a smaller region (the exit region) which can be achieved reliably from any point in the landmark region. This landmark definition restricts the information considered by the planner even further, but avoids the need for infinitely precise localization, which is realistically impossible to satisfy.

7.5.1 Computing the landmark region

During navigation, an on-board processor processes images iteratively. Each image is acquired as a 260×240 frame. The detection algorithm detects a landmark only if the outer circle is almost completely visible. Since the orientation of the camera relative to the landmark pattern can be arbitrary, a specific landmark induces a circular area C in the workspace from which the pattern is guaranteed to be entirely visible. Fortuitous orientations of the camera may allow the landmark to be detected outside

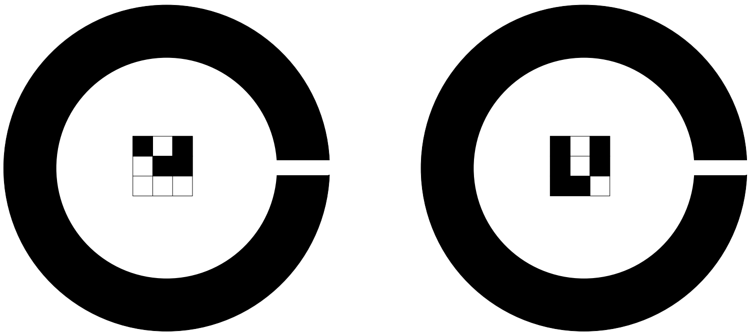


Figure 7.7: Two possible "ceiling" landmarks

this area, but this does not affect the validity of a plan generated by our planner (although it may slightly speed up execution).

According to the above, landmark regions are created by finding the radius r_{max} of the biggest disk that can be inscribed in the viewing area of the camera (projected on the ceiling), and then drawing a disk centered at each landmark with radius $r_{max} - r_l$, where r_l is the radius of the outer circle of the landmark. Clearly, for landmark regions to exist, r_{max} must be bigger than the radius of the outer circle of the landmarks.

Non-instantaneous detection may still cause a landmark to be missed. If the robot follows a path almost tangent to C, it may traverse and exit C while analyzing an image acquired just before entering C. This leads us to define the actual landmark region L concentric to C and ε-smaller in radius. The difference ε is derived from the maximal velocity of the robot and the time necessary to analyze an image, so that, if the robot follows a path tangent to L, it is guaranteed to acquire and process an image while still in C.

7.5.2 Recognition and localization

The 260×240 camera frame is obtained as a gray-level image and it is binarized using an adaptive threshold. Connected components in the binary image are identified, and their size is compared to the expected size of the landmarks. Each component whose size is similar to the landmark size is compared pixel-by-pixel to a model of the landmark outer circle. A sufficiently low number of pixel mismatches indicates that a landmark has been found. Once the landmark has been detected, the opening of the outer circle is used to compute the orientation of the landmark relative to the robot. The inside grid is used to uniquely identify the landmark. From this information (and the landmark coordinates in the workspace frame) it is straightforward to compute the position and orientation of the robot in the workspace frame of reference.

Localization assumes that the landmark patterns are horizontal and the camera axis vertical. Deviations from these orientations cause errors. The ceiling of our laboratory is sufficiently horizontal, but the floor is rather bumpy and sometimes has noticeable local slopes. In order to get the best possible localization accuracy, we must avoid placing landmarks over such uneven areas of the floor. After several measurements, the experimental bound of the localization error was found to be $\pm 3°$ for the orientation and 0.22 inches for the position.

Adding the 1° control uncertainty to the orientation error, and making sure that the initial aiming uncertainty of the robot is less than 3°, we may use $\theta = 4°$.

The radius η of the exit region for each landmark must be bigger than the position error. Our experience from the previous set of experiments prevents us from choosing η too close to 0.22 inches, because in this case the achievement of the exit region would be difficult, and it would require several attempts. Thus, we choose $\eta = 0.5$. This conservative bound also takes into account the small error in positioning the landmark relative to the workspace and the fact that the floor is not perfectly parallel to the ceiling (causing the camera to tilt slightly relative to the landmark). Theoretically, we may put an arbitrary number of exit regions in each landmark region. For simplicity, we use only one exit region per landmark, concentric with the landmark region. (Localization precision is better near the center of the landmark region.)

The reliability of detection and recognition is affected by the proximity of the landmarks to light sources. (Our laboratory is illuminated by ceiling fluorescents.) With no light sources adjacent to a landmark (so that the landmark region does not overlap a light source), detection and recognition are 100% reliable.

The landmarks have an outside diameter of 10 inches and are located about 8 feet above the camera. The image of a landmark has a diameter of 92 pixels. The diameter of the landmark region induced by a landmark is 16.2 inches. With the hardware available on NOMAD-200, image processing takes about 0.6 seconds. A robot moving at 4 inches per second will move 2.4 inches during one cycle of image processing. A landmark diameter of 16 inches is sufficient to guarantee that the robot will never miss a landmark.

7.5.3 More engineering

There are several factors that constrain the placement of landmarks in the workspace. First, they must not be adjacent to light sources. Second, they must not be over uneven areas of the workspace floor. Third, the currently implemented planner requires that landmark regions do not intersect C-obstacles. Obeying these constraints makes it impossible to place enough landmarks in our laboratory workspace for reliable navigation. In fact, using $\theta = 4^\circ$, the planner returns failure for the configuration of Figure 7.8, because the robot cannot get safely through the narrow corridor at the left. (There is a light fixture at the center of the corridor that prevents us from placing a landmark there.) Using a smaller value of θ results in plans, but their execution is unreliable; sometimes, the robot collides with the corridor walls.

To deal with this problem we use a refined version of the image analysis techniques to compute the robot's orientation with greater accuracy (since this is the main cause of directional uncertainty). By extracting contours at a sub-pixel resolution we are able to reduce θ from 4° to 2°, which is sufficient to navigate reliably through the corridor. Even smaller values could be possible by designing a landmark pattern containing a better orientational feature or by equipping the camera with a zoom to get a higher-resolution image of the landmark, once it has been detected.

Extracting contours at sub-pixel resolution typically requires more computation time than the techniques presented in Subsection 7.5.2. Although this does not affect detection or recognition (therefore the size of the landmark regions is not affected), it requires that the robot spend more time within the landmark region performing localization. Since we wish to avoid this delay if it is not necessary, we use two values of θ, θ_1 (the directional uncertainty with the simpler image analysis technique) and θ_2 (the uncertainty with the more sophisticated technique). The planner uses θ_1 until it fails (if this eventually happens). If an iteration fails (i.e., no new landmark region can be added to the goal extension), it is repeated with θ_2. If it is then successful, the planner shifts back to θ_1 at the next iteration, and so on. The planner associates the value of control uncertainty, θ_1 or θ_2, to every command associated with a landmark region. At navigation time, this value is used to decide whether the more sophisticated image analysis techniques must be used, or not.

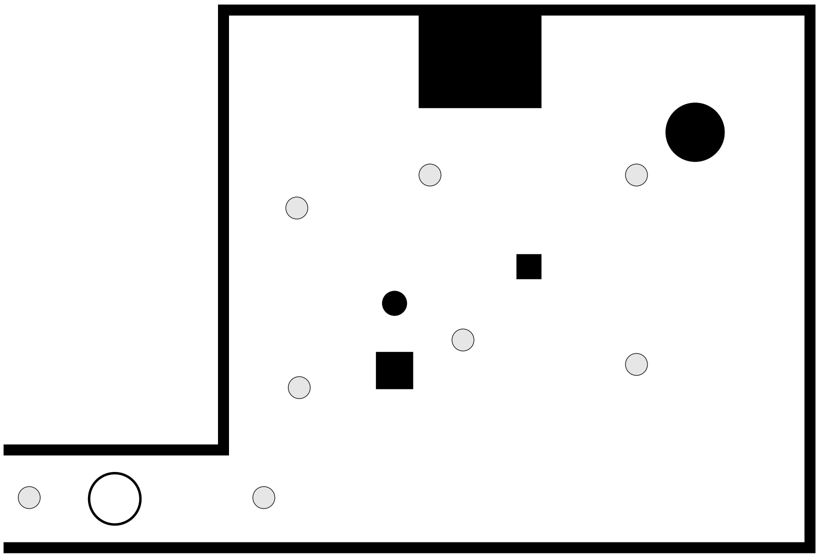
This approach can be easily extended to more than two values of control uncertainty, or even to a continuous dependence of uncertainty upon image resolution, by using a multi-parametric planning algorithm (see Chapter 6). Thus, we can derive guaranteed plans which at each step use the maximum possible uncertainty value, i.e., the lowest possible resolution, and, consequently, devote the least possible time to the localization procedure.

7.5.4 Plan execution

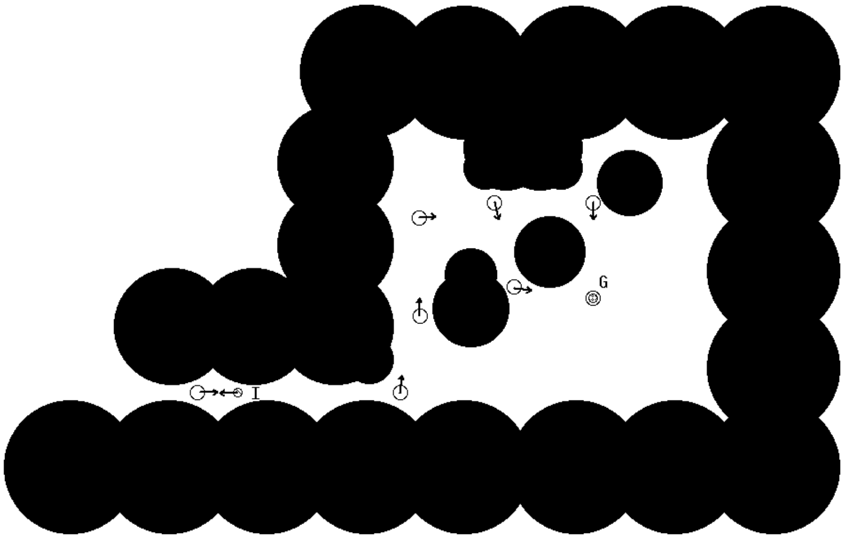
Figure 7.8(a) shows a map of our laboratory. It consists of a narrow corridor leading into an open space populated with landmarks (gray) and obstacles (black). The white disk represents the robot, which has a diameter of 24 inches. Figure 7.8(b) shows the model of the configuration space input to the planner. C-obstacles are represented as a collection of black disks. The landmark regions are represented as small white disks with arrows attached to them. The arrows show the direction of motion that the robot should follow once it gets in the exit region of the corresponding landmark. One of the landmark regions completely contains the goal **G** of the robot. No arrow is attached to this landmark region. The robot begins its motion from some point in the small disk marked with **I** in the corridor.

Initially, the robot backtracks in the corridor to get into the leftmost landmark region and obtain an accurate measurement of its position using sub-pixel resolution. Then, it moves along the corridor into the open space, and, hopping from (landmark) region to region, eventually gets into the goal. It should be noted that it is not possible for the robot to move directly into the open space, because its initial uncertainty is too high and it may hit a wall. The robot has to get into the landmark region first, in order to reduce its position uncertainty.

We executed the above plan numerous times and it never failed. In fact, once the robot reached its goal, we randomly selected a different landmark as its new



(a) the laboratory space



(b) the workspace model provided to the planner

Figure 7.8: The experiment with ceiling landmarks

goal, replanned and executed again. We repeated the above procedure over and over until the robot ran out of power. Our system was able to navigate the robot reliably between landmark regions for the entire battery life of the robot (about one hour).

7.6 Evaluation

Experimentation always teaches that there are more sources of uncertainty than one can anticipate. Our experiments are no exception. Nevertheless, through iterative engineering of the workspace, we were able to eliminate all sources of uncertainty that do not fit into our uncertainty model, and verify the successful execution of the resulting plans. The major asset of our method is that the uncertainty of the robot does not deteriorate over time (as when using odometric sensors), but remains bounded by the localization error. A robotic system using our planning-navigation method can operate with no problems over long periods of time. Furthermore, the system has a fast response to requested tasks, because planning time is a polynomial function of the complexity of the workspace, and it can deal with unexpected events gracefully. The major problem with our method is that it solves only a fraction of the planning problems, that would be solvable if more information were used by the planner. This problem can be addressed with the placement of more landmarks in the workspace, or with the use of hierarchical landmarks.

In what follows, we comment on the performance of our system with respect to the criteria we established at the beginning of this chapter.

- **Cost:** The landmarks in both sets of experiments are very cheap and easy to build. The major cost comes from the sensing system that requires the use of a camera, a frame-grabber, and an image processor.
- **Completeness:** Our planner would be able to solve many more problems, if its implemented version could handle overlapping generalized polygonal landmark and C-obstacle regions. Covering the obstacles with circular disks unnecessarily consumes free space (or, increases the complexity of the workspace). Also, not being able to have overlapping obstacle and landmark regions places a significant

constraint on the location of the landmarks. After the implementation of these two features, both problems will disappear.

A more important completeness issue is that our system is limited by the accuracy of the localization through landmarks, and it cannot achieve goal regions that do not overlap landmarks, something that NOMAD can do by use of its odometric sensors. However, nothing prevents the execution module from using more information than that considered by the planner. Furthermore, we can introduce a hierarchical structure of landmarks (easily recognizable landmarks for general navigation at relatively high speeds, and more subtle landmarks for more detailed tasks) to enhance the problem solving capabilities of our method.

- **Correctness:**

  Using the standard landmark definition we are not able to produce correct plans, because we cannot possibly eliminate position uncertainty in the landmark regions. The only thing we can do is to verify correctness after the creation of the plans. Using the generalized landmark definition in the experiments with the ceiling landmarks and some further engineering of landmark placement, we are able to produce plans that succeed every time they are executed.

- **Robustness:** This is an area where our system performs really well. Whenever something unexpected happens, the robot attempts to detect some landmark (if necessary, by moving randomly in the workspace). If successful, it calls the localization algorithm and gets an estimate and an error bound of its current position. Using this information as a new initial region, it either decides that it is in some landmark region from where it knows how to get to the goal, or it replans. Replanning is very fast, because a plan associated with many landmark disks already exists. For the recognition of unexpected events we can use other sensors of NOMAD, like odometric sensors to detect moving too far beyond the landmarks in the termination condition, bumpers to detect collisions, etc.

Several other practical mobile robot systems have been built and tested successfully [44, 73]. Our system is different in that it combines polynomial planning with reliability under uncertainty. The cost we pay is the effort to engineer the workspace, and the opportunity cost of not using all information available to the robot. We have shown that this tradeoff is acceptable. Our system is still in its infancy and requires further development and rigorous testing.

Chapter 8

Conclusions

8.1 Summary

In this thesis we have addressed the problem of dealing with uncertainty in robot motion planning. This problem has been, up to now, a major obstacle for the widespread use of robots in practical applications. The various formulations of the problem that have been proposed result in planning algorithms of exponential complexity and, hence, of no practical use. Another school of thought proposes not to reason about uncertainty at all. Instead, it proposes to equip the robot with primitive heuristic behaviors so as to help it deal with unexpected events at execution time. Although fairly reliable and robust robotic systems of this kind have been built, their plan execution often lacks deliberation and can be sidetracked by local error-coping behaviors. Even worse, when such plans fail, they do not provide useful feedback about their failure.

Our work begins with the premise that intelligent agents should be deliberative during plan execution. Therefore, they should be able to predict as many errors as possible, so that their error-coping behavior will be dictated by the drive to achieve the ultimate goal of the plan. Such plans require reasoning about uncertainty at planning time. We also believe that planning should be fast; hence, we cannot use uncertainty models that result in intractable problems. In Chapters 4, 5, and 6 we describe an uncertainty model that admits a correct, complete, and polynomial planning algorithm. We discuss several planning examples solved with an implemented version of the algorithm. Next, we examine whether it is physically and financially feasible to engineer the system of a robot and its workspace, so that the assumptions underlying the model are satisfied. By making sure that the planning algorithm is provably correct, the role of experimentation shifts from validating the algorithm to validating the assumptions on which the algorithm depends. Experimental failures can be easily tracked down to discrepancies between the model and the real world, and further engineering can be utilized to minimize these discrepancies. Thus, workspace engineering becomes an essential ingredient of problem solving. In Chapter 7 we show how this process actually leads us to the design of a cheap reliable robot navigation system.

Typically, uncertainty grows during navigation. To design a system that does not fail over long periods of navigation, some kind of information must be extracted from the environment. As information does not come without errors, reasoning about it adds complexity to the planning process. An efficient planner must carefully select the pieces of information it takes into account. In our model, the robot gains information about the workspace through special features called landmarks. All the points of the workspace from where a landmark is perceivable are considered equivalent from the planner's point of view. Therefore, the world can be described by a finite number of states. By assuming that landmarks are individually identifiable by the robot, we avoid having to consider combinations of states, and we preserve the polynomiality of the world description. Whenever the robot can sense a landmark, it is able to identify the state of the world without error. A guaranteed plan consists of motion commands that always reveal at least one landmark when executed. Thus, planning is reduced to computing the connectivity (with respect to the available motion commands) of the graph representing the world states, and then searching this graph.

Computing the connectivity of the world states is performed with the help of the preimage backchaining method. The preimage of a goal is a structure which depends on the planning parameters (commanded direction of motion, directional uncertainty, etc.), and which can be used to determine whether a motion command is guaranteed to succeed or not. We prove that we need not search the entire continuous parameter space for guaranteed commands. Instead, it is possible to divide the parameter space into a polynomial number of cells. All points in a cell have the same properties (with respect to planning) and, thus, only one point per cell need be considered.

We implemented one such planner in two dimensions for one control parameter (commanded direction of motion) and two control parameters (commanded direction of motion and directional uncertainty). Both planners are polynomial in the complexity of the workspace. We then attempted to create an actual workspace-robot combination for which our assumptions are satisfied to a large extent. By placing black-and-white patterns on the ceiling of the workspace and having a camera mounted on the robot look straight up, we were able to satisfy most of the assumptions. Several sources of errors caused failures, but we were able to eliminate all but one with recursive workspace engineering. The one unrealistic assumption is the requirement that the robot have perfect position sensing whenever it enters a landmark region. To overcome this difficulty, we adapted our model to allow for some uncertainty. We were able to do so with the notion of generalized landmarks, where perfect position sensing is not required. The resulting system was able to repeatedly operate without failures for the duration of the battery life of the robot (about one hour).

Our experimental work has validated every single aspect of the proposed planning paradigm. By working with a correct algorithm, experimental failures were easily tracked down to discrepancies between the world model and the real world. Both engineering and model refinement were used to eliminate these discrepancies. The cost of engineering the workspace was minimal, and the reliability and robustness of the resulting system were high.

8.2 Limitations of the approach

Any new piece of work, especially one that proposes an alternative perspective of a hard problem, is bound to draw some fire. In addition, computer scientists are traditionally suspicious of polynomial algorithms that are used for intractable problems.

So it is no wonder that our work has attracted some criticism. In this section we attempt to answer some of the questions that have been raised about our work, and also try to evaluate its strengths and shortcomings.

Is the model realistic? One concern that has been voiced about our work is that its assumptions are not realistic. This is not the point. Our approach is not intended for workspaces over which we have no control. It is supposed to be used in environments where we can intervene and make adjustments. We begin from a set of assumptions that produce a polynomial algorithm, and then we engineer the workspace to satisfy our assumptions. Therefore, the question becomes whether such engineering is possible and at what cost. Experimentation indicates that it is possible to enforce the assumptions at a reasonable cost.

Is the model simplistic? Another criticism has been that, by considering only a small subset of the available information, our planner is not able to produce plans that are within the capabilities of the robot. This is true. To obtain, however, polynomial planning algorithms, it is necessary to simplify the information that is considered by the planner. Several attempts to use more complex information structures have yielded exponential algorithms. Also, information that is being ignored at planning time may still be used during plan execution. We believe that it is of utmost importance that practical robotic systems have the capability of fast reliable planning. If such planning fails, other more complex algorithms can be called to solve the problem.

What about model error? Small errors in the location of obstacles can be dealt with by slightly growing the obstacles. Small errors in the location of landmarks can be dealt with by shrinking the landmark region and growing the localization uncertainty within landmark regions. These adjustments decrease the plan-finding capabilities of our planner, but preserve its polynomiality. We believe that an important target of future research should be to augment our planning model to account for model error without loss of polynomiality. We conjecture that such an extension is possible.

What about unexpected events? No planning system can cope with arbitrary unexpected events. The term "unexpected event" (in the context of motion planning) usually refers to gross discrepancies between the workspace model and the actual world, and, as such, it is a form of model error. Our system is not designed to reason about such situations, because it depends on prior knowledge of the planning environment. Since unexpected events cannot be completely eliminated, a realistic robotic system should be equipped with the capability to deal with them effectively. In Chapter 5, we mention a case where a robot can cope with one kind of unexpected events (stationary unexpected obstacles that partially block the robot's forward projection) without abandoning its current plan. However, a complete robotic system should be able to cope with more disrupting events, like moving obstacles, complete blockage of passage, disappearance of landmarks, etc. In this case, local heuristic behaviors will be temporarily necessary. After the emergency is over, the robot should attempt to locate a landmark and reattach to the current plan. Reattachment is easy because of two important properties of the plans produced by our system, namely, that they are distributed over the workspace, and that their execution need not be sequential. A plan is simply a collection of motion commands, each attached to a landmark area. Each motion command is guaranteed to lead the robot either to the goal or to another landmark area with a similar motion command attached to it. In fact, the plans can be constructed so that a motion command is attached to all landmark areas from where a guaranteed motion plan (with respect to a specific goal) exists. Thus, if the normal execution of a plan is disrupted, the chance of reattachment is maximal. Hence, our system is robust in dealing with unexpected failures.

Why is this approach better? Several other practical robotic systems have been designed and tested successfully ([44, 10]). However, no other system combines polynomiality of planning and deliberation during execution. Our simple experiments validate our problem solving methodology, and demonstrate that fairly simple workspace engineering is sufficient for the creation of a reliable navigation system. By no means do we claim that our system is ready for real-world applications. It should be augmented with capabilities like automatic extraction and update of the workspace model and dealing with unexpected events. But the major problem, dealing with uncertainty in an efficient way, has been solved.

8.3 Further research

Several interesting problems warrant further research. Here we discuss the ones that we consider as the most important.

Complexity issues In Section 4.6 we compute the computational complexity of the monoparametric planning algorithm. If s is the number of maximal connected landmark regions and l is the complexity of landmark and C-obstacle regions, we prove that the complexity is in $O(sl^3 \log l)$. We also mention that runs of the implemented algorithm indicate that a tighter bound may exist, and we outline how one could attempt to obtain an $O(sl^2 \log l)$ bound. Two open problems that need to be addressed before such a bound can be obtained are (a) to prove that the number of O-Spike events during one iteration of the algorithm is quadratic instead of cubic, and (b) to prove that the total number of topological changes of the preimage during catastrophic events is also quadratic. We conjecture that both proofs are possible with the use of techniques similar to the ones used in [8].

Extension to three dimensions The algorithms we developed apply to motions in two dimensions. In Subsection 6.4.6, we show how to apply them in 3-D space, if motions along the third dimension and motions in the plane are separated. It would be interesting, and useful, to see whether our ideas can be extended to 3-D translational motions, or to planar motions that combine simultaneous translation and rotation. In the latter case, the robot is not confined within a single slice of the nondirectional preimage during the execution of a motion command, but it can move across slices. A motion command needs a number of parameters (not only one) to be completely determined. The number of parameters depends on the available controls of the robot's motion. Similarly, in 3-D translation the directional preimage is a 3-D structure and the robot is free to move in all three dimensions. A motion command can be fully determined with a direction in 3-D space (i.e., two parameters are needed). If appropriate uncertainty models are used, both problems can be probably solved with the multiparametric analysis of Chapter 6.

Other models of uncertainty accepting polynomial planners Our model is not the only one that admits a polynomial planner. It is certainly worthwhile developing other models that lead to polynomial planning algorithms. Different models may be useful under different circumstances.

Probabilistically guaranteed plans and EDR plans Our planners produce guaranteed plans. However, in many realistic situations guaranteed plans do not exist. In Section 6.3, we describe in detail how our methodology can be used to produce probabilistic plans with maximum likelihood of success. Unfortunately such plans may fail in unacceptable ways. In Subsection 6.3.3, we discuss plans the fail recognizably (EDR plans), and in Section 5.7 we discuss plans that are probabilistically guaranteed. Although we present algorithms for the computation of both kinds of plans, we do not perform any complexity analysis. It is very interesting to investigate whether EDR and probabilistically guaranteed plans can be produced within the landmark framework in polynomial time. Our conjecture is that exponential time is indeed necessary. If this is the case, it would be instructive to identify further adjustments to the uncertainty model that would render planning polynomial.

Application to other domains Our problem solving methodology does not specifically apply to robot motion planning with uncertainty. Many applications that have to deal with the complexity of the physical world may benefit from the approach of artificially decreasing the physical complexity of a system, in order to develop problem solvers with sufficiently low computational complexity.

Appendix A

Properties of the Locus of a Spike

In this appendix we establish the equation of the locus of a spike and we compute 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, E-Spike-In, E-Spike-Out).

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. The angle of the rays is constant and equal to 2θ (the directional uncertainty), and its bisector is 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. We also develop formulas to identify the valid part of the locus curve. (A point of the locus is not valid, if one of the rays that form the spike at this point crosses through a landmark or obstacle disk.)

Let us consider the case where the intersecting rays are tangent to two distinct landmark disks D_1 and D_2 of respective radii η_1 and η_2. One ray, l_1, is the left ray of D_1; the other, r_2, is the right ray of D_2, as shown in Figure A.1. 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 $-\eta_i$. If the spike is produced by a single landmark disk, then its locus is simply a circle having the same

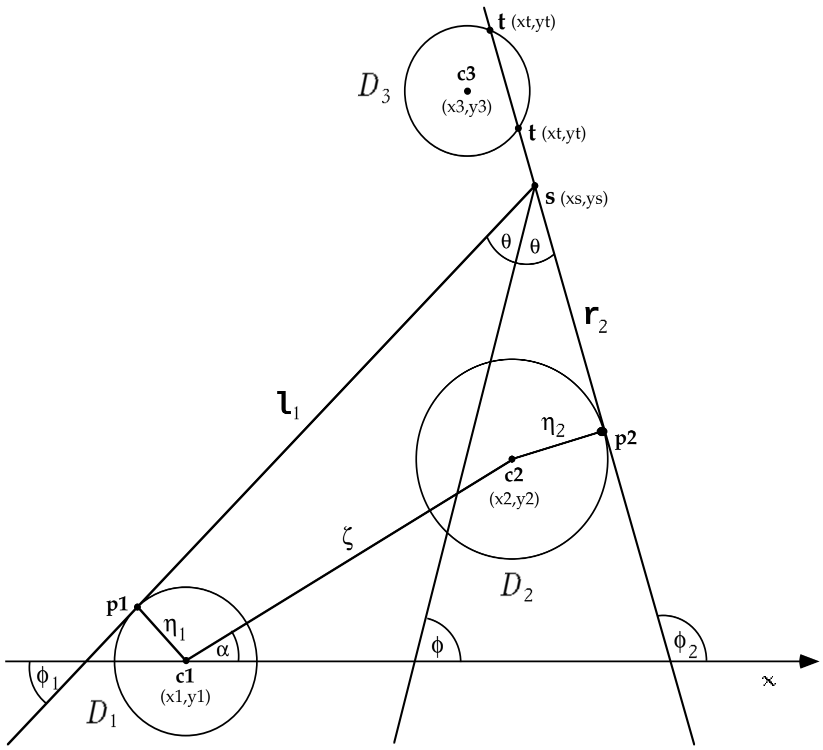


Figure A.1: A spike created by two landmark disks

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 D_1 (resp. D_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 (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:

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

Both equations must be verified for $x = x_s$ and $y = y_s$, yielding the following

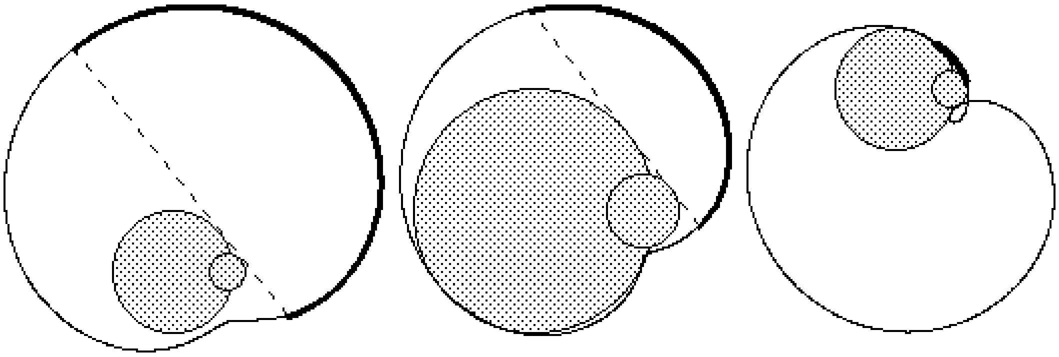


Figure A.2: Various shapes of spike-locus curves

parametric equations of the spike-locus curve:

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

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 Equations (A.1) and (A.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, depending on the sign 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 Figure A.2, for two intersecting landmark 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 the bigger disk.
- 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 Figure A.2, $\eta_1 = 50$, $\eta_2 = 15$, and $\theta = .25$; both λ_1 and λ_2 are

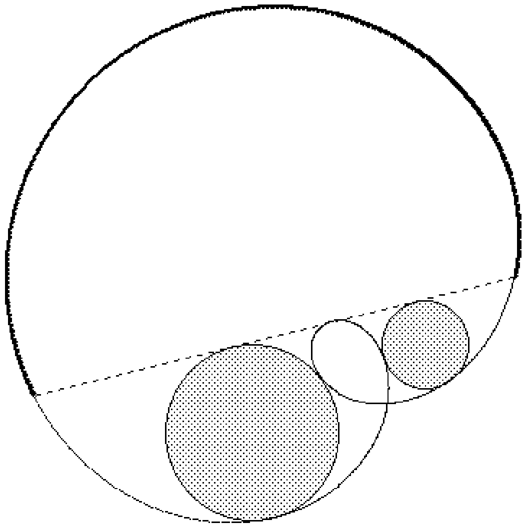


Figure A.3: Spike-locus curve for two disjoint landmark disks

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 quantities 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 Figure A.3.

Notice that not all points in a spike-locus curve correspond to feasible spikes. Let us draw the line tangent to both D_1 and D_2, and oriented so that it touches D_1 before D_2 (the dashed line in Figure A.2). The valid part of the locus (shown in thicker line in the figure) lies on the left-hand side of this line. Intersecting a spike-locus curve with a circle may yield both valid and invalid points, which must be subsequently classified accordingly.

Intersection of a Spike-Locus Curve with a Circle

Let us now consider a third disk D_3 centered at c_3 and having radius η_3 (see Figure A.1). We wish to compute the intersection of the spike-locus curve with the circle $\mathcal{C}_3$ bounding this disk. To that end, we denote any intersection point of r_2 with $\mathcal{C}_3$ 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 Equations (A.1) and (A.2), we get (after some calculation):

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

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 $\mathcal{C}_3$ 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 $\mathcal{C}_3$. After yet some calculation, we find:

$$\begin{aligned} x_t \;&=\; x_3\cos^2\varphi_2 + x_2\sin^2\varphi_2 + \eta_2\sin\varphi_2 - \\ &\qquad (y_2 - y_3)\sin\varphi_2\cos\varphi_2 \pm \cos\varphi_2\sqrt{\eta_3^2 - B^2}, \qquad (A.4)\\ y_t \;&=\; (x_2 + x_3)\cos\varphi_2\sin\varphi_2 - \eta_2\cos\varphi_2 + \\ &\qquad y_3\sin^2\varphi_2 + y_2\cos^2\varphi_2 \pm \sin\varphi_2\sqrt{\eta_3^2 - B^2}, \qquad (A.5)\end{aligned}$$

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 Equations (A.4) and (A.5), we get:

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

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

Comparing Equations (A.3) and (A.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:

$$S_0 + S_1\sin\varphi + S_2\cos\varphi + S_3\sin^2\varphi + S_4\cos^2\varphi + S_5\sin\varphi\cos\varphi \;=\; 0, \tag{A.7}$$

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 $u = \tan(\varphi/2)$, Equation (A.7) becomes:

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

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

Classification of Solutions

As we mentioned above, not every real solution of Equation (A.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 moves 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 D_1 and D_2 (see Figure A.4). 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:

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

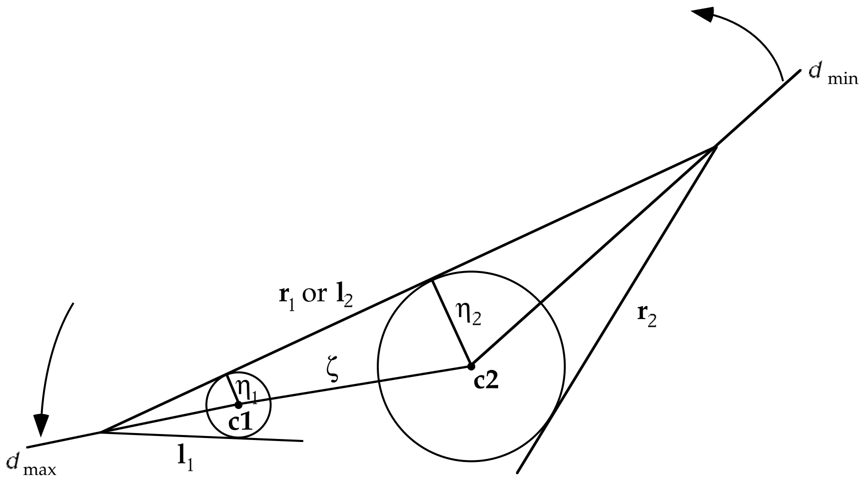


Figure A.4: Spike valid limits

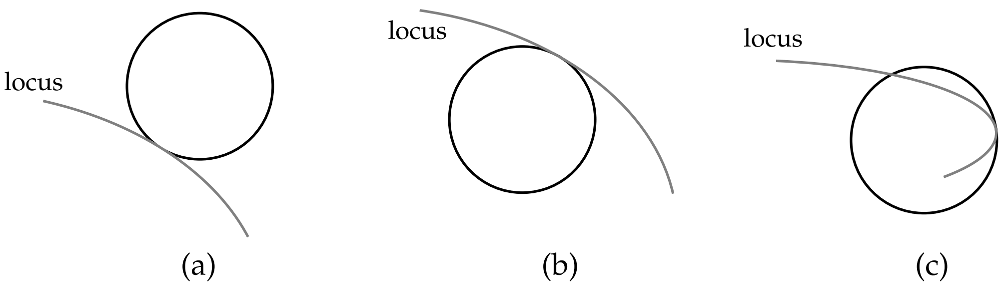


Figure A.5: Different tangent positions

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

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 $\mathcal{C}_3$, 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 $\mathcal{C}_3$ are tangent and exterior to each other (see Figure A.5 (a)). If $\varphi_s = \varphi_d$ the two curves are tangent with one of them lying inside the other (see Figure A.5 (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 $x_s'' = d^2 x_s / d\varphi^2$ and $y_s'' = d^2 y_s / d\varphi^2$. If $\eta_s > \eta_3$ the spike locus encloses $\mathcal{C}_3$ (Figure A.5 (b)); if $\eta_s < \eta_d$ the spike locus is enclosed by $\mathcal{C}_3$ (Figure A.5 (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 $\mathcal{C}_3$. However, the study of this case is of interest if we wish to compute the critical values of θ where the envelope of a non-directional preimage becomes tangent to a disk.)

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. 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:

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

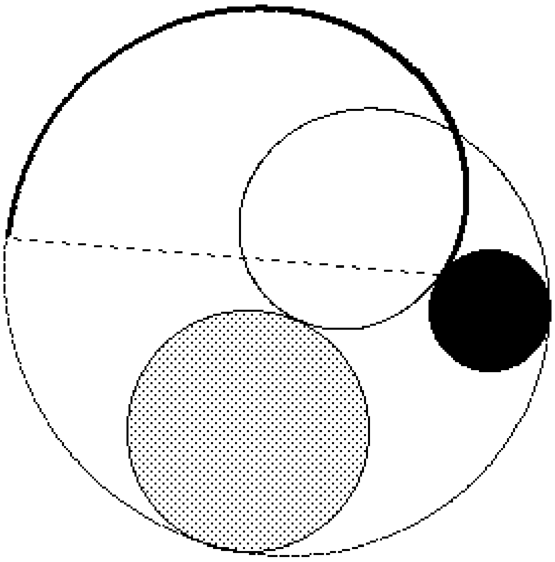


Figure A.6: Spike-locus curve for disjoint landmark and obstacle disks

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

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

and for an obstacle-obstacle spike:

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

Regarding the shape of the spike-locus curve, it still depends 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 has an inner loop (see Figure A.6).

Let us consider now the case when the two disks intersect with each other. (See Section 5.3 for a description of this case.) Again we begin by examining a landmark-obstacle spike. (See Figure A.7 for illustration.) The termination of this spike cannot

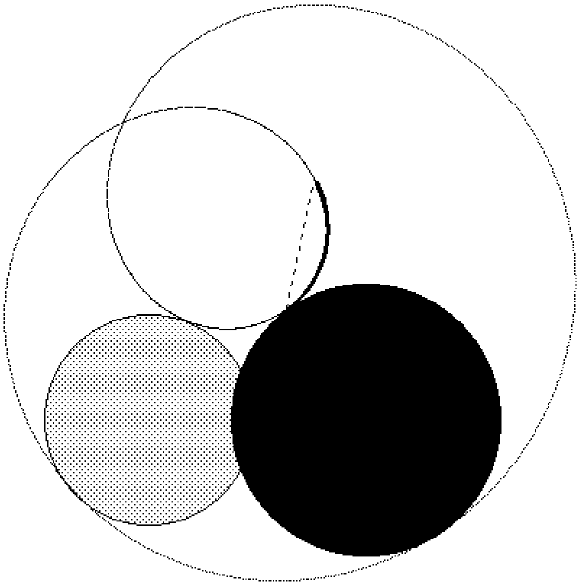


Figure A.7: Spike-locus curve for intersecting landmark and obstacle disks

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 *E-Right-Anchor* event), it sticks there for a while as the ray continues to rotate counterclockwisely. When the ray gets tangent to the landmark disk (an *E-Right-Release* event), it turns into a landmark ray and its origin starts moving in the boundary of the landmark disk. Thus, the landmark-obstacle spike is transformed, first into a *right-anchored spike*, and then into a landmark-landmark spike. Since the right-anchored spike has a different equation, the termination of the original landmark-obstacle spike occurs exactly at the E-Right-Anchor event. The calculation of the E-Right-Anchor 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:

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

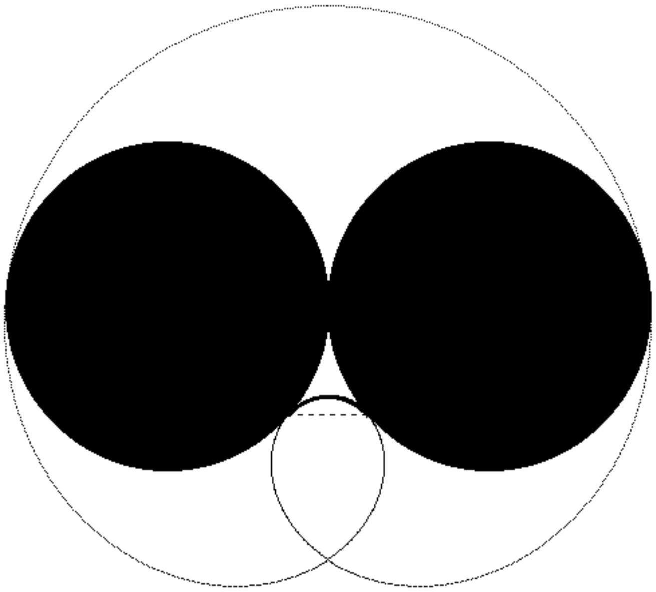


Figure A.8: Spike-locus curve for two intersecting obstacle disks

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

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

Two intersecting obstacle disks may create a spike, but this spike can only occur "under" the obstacles (see Figure A.8). 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:

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

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:

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

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

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

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

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

Combining the relations (A.11) and (A.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 (A.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.

Appendix B

Properties of the Spike Curve

Definition

A *spike curve* represents the relationship between the orientation of the commanded velocity vector d, and the half-angle of the uncertainty cone θ, such that the spike of two disks D_1 and D_2 lies on the boundary of a third disk D_3. The right ray (stemming from D_2) has slope $d + \theta + \pi$.

Parametric equations of the spike curve and its slope

Notice that we can find all points of the spike curve that corresponds to these three disks, by having a point p move on the boundary of D_3, and from every such position draw tangents to D_1 and D_2. If the slopes of these tangents are s_1 and s_2 respectively, then $d = (s_1 + s_2)/2 + \pi$, and $\theta = (s_2 - s_1)/2$. Now remember that each point on a spike curve corresponds to an L-Spike critical event, where the rays forming the spike just intersect an intermediate-goal disk. Therefore, there should exist another point p' lying arbitrarily close to p, such that, if we draw the tangents to D_1 and D_2 from it, they do not intersect D_3. This can only be true, if both rays drawn from p leave D_3 immediately, without traversing its interior. Figure B.1 presents one acceptable and one unacceptable case. We are only interested in the positions of p on the boundary of D_3, which lie between the tangency points of the appropriate common tangents between D_3 and the other two disks. This suggests that we can track the spike curve, if we allow p to move counterclockwisely on the boundary of D_3 from its leftmost to

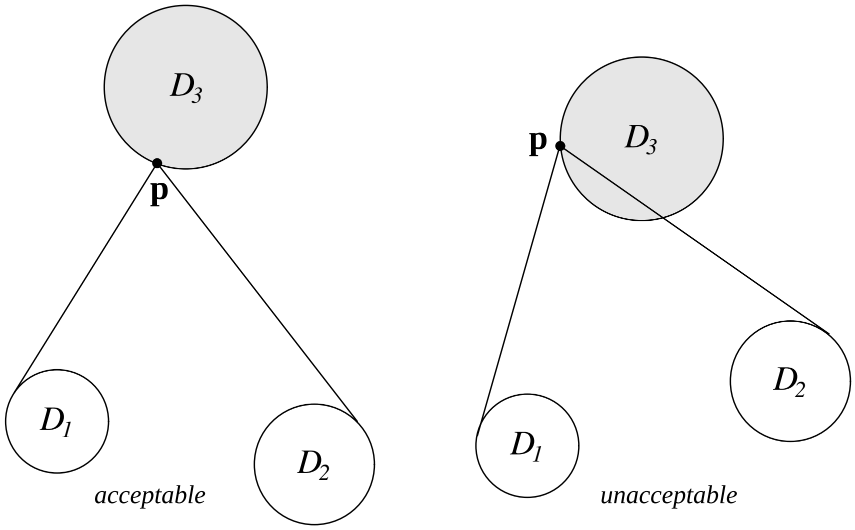


Figure B.1: Rays forming a spike should not cut through D_3.

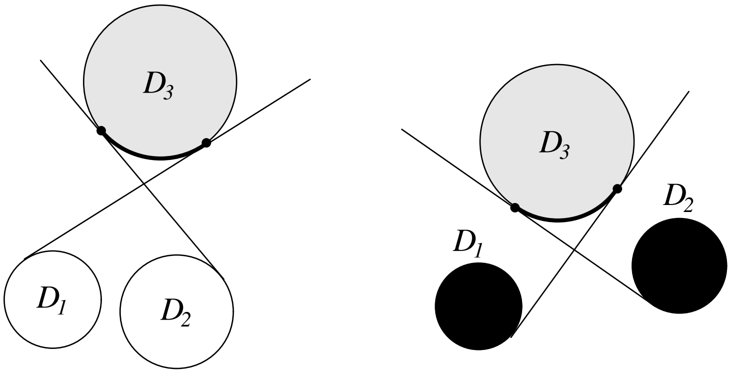


Figure B.2: Limits of motion of p on D_3.

its rightmost point, and compute d and θ for each location. In Figure B.2 we show the valid locations of p in two cases, when both D_1 and D_2 are landmark disks, and when both are obstacle disks.

In the following we will assume that D_1 and D_2 are landmark disks. However, it is easy to adapt the equations and the proofs for the cases, where either or both are obstacle disks: In each formula or equation we just need to switch the sign of the radii of obstacle disks, wherever they appear.

Look at Figure B.3. The parameter we use to represent the location of p on the boundary of D_3 is the angle x between the line connecting the center c_3 of D_3 and

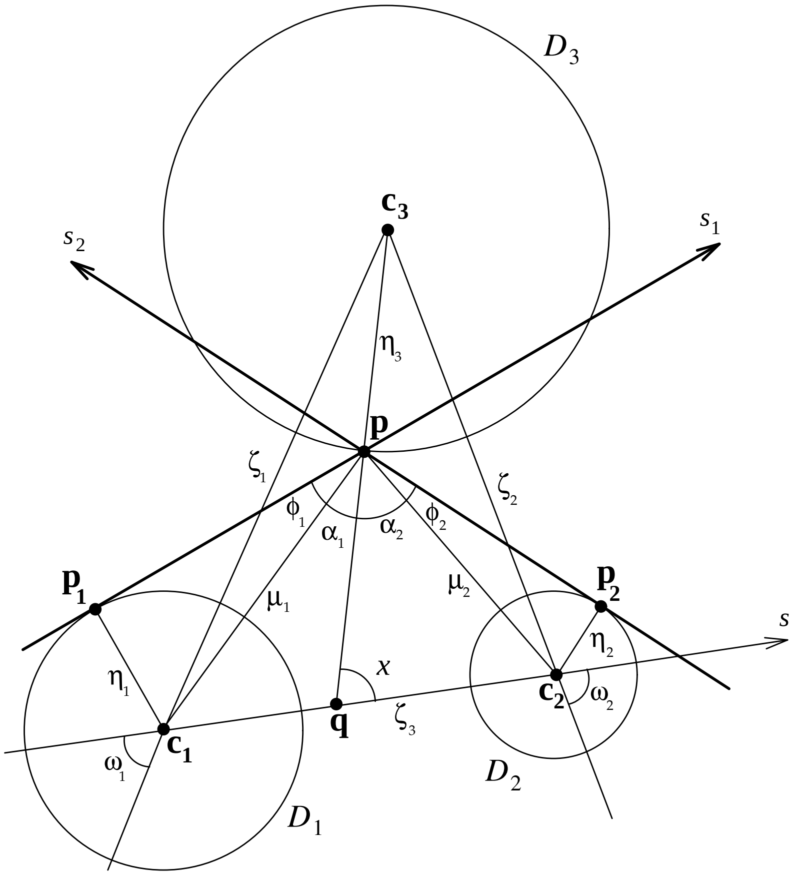


Figure B.3: Spike created by two landmark disks

p and the line connecting the centers c_1 and c_2 of D_1 and D_2. Let the radii of the three disks be respectively η_1, η_2 and η_3; let the distance between c_1 and c_3 be called ζ_1, between c_2 and c_3 ζ_2, and between c_1 and c_2 ζ_3; let the distance between p and c_1 be called μ_1, and between p and c_2 μ_2. Let the point of intersection of the lines that connect c_3 with p and c_1 with c_2 be called q. Also, let the points where the two rays are tangent to D_1 and D_2 respectively be called p_1 and p_2. We call angle $\widehat{p_1pc_1}$ φ_1, angle $\widehat{c_2pp_2}$ φ_2, angle $\widehat{c_1pq}$ α_1, angle $\widehat{qpc_2}$ α_2, angle $\widehat{c_3c_1c_2}$ ω_1 and angle $\widehat{c_3c_2c_1}$ ω_2. Finally, let the slope of the line that connects c_1 with c_2 be s. It is easy to see that:

$$\theta = (\varphi_1 + \varphi_2)/2 + (\alpha_1 + \alpha_2)/2 \tag{B.1}$$

$$d = (\varphi_2 - \varphi_1)/2 + (\alpha_2 - \alpha_1)/2 + x + s + \pi \tag{B.2}$$

We will use the above equations to find a parametric representation $(\theta(x), d(x))$ of the spike curve. The angle x is a suitable parameter, because it grows monotonically as p moves from its leftmost to its rightmost position. We therefore need to express φ_1, φ_2, α_1 and α_2 as functions of x. We can do this in two steps: first express them as functions of μ_1, μ_2 and x, and then find the dependence of μ_1 and μ_2 on x, as the following equations show:

$$\begin{aligned}
\sin\varphi_1 &= \eta_1/\mu_1 && \text{(B.3)}\\
\sin\varphi_2 &= \eta_2/\mu_2 && \text{(B.4)}\\
\sin\alpha_1 &= \zeta_1 \sin(x-\omega_1)/\mu_1 && \text{(B.5)}\\
\cos\alpha_1 &= (\zeta_1 \cos(x-\omega_1) - \eta_3)/\mu_1 && \text{(B.6)}\\
\sin\alpha_2 &= \zeta_2 \sin(x+\omega_2)/\mu_2 && \text{(B.7)}\\
\cos\alpha_2 &= (-\zeta_2 \cos(x+\omega_2) - \eta_3)/\mu_2 && \text{(B.8)}\\
{\mu_1}^2 &= {\zeta_1}^2 + {\eta_3}^2 - 2\zeta_1\eta_3\cos(x-\omega_1) && \text{(B.9)}\\
{\mu_2}^2 &= {\zeta_2}^2 + {\eta_3}^2 + 2\zeta_2\eta_3\cos(x+\omega_2) && \text{(B.10)}
\end{aligned}$$

For the angles φ_1 and φ_2 we use only one equation because we know that they are always positive, whereas for α_1 and α_2 we need a second equation for each, in order to determine their sign. The above equations combined with the equations for θ and d define the parametric form of the spike curve. The parameter x varies from a minimum value x_0 that corresponds to the rightmost location of p, to a maximum value x_1 that corresponds to the leftmost location of p. In the first case the left ray is tangent to D_3, and in the second case the right ray is tangent to D_3. If both D_1 and D_2 are landmark disks we have:

$$\begin{aligned}
x_0 &= \omega_1 + \arccos\frac{\eta_1+\eta_3}{\zeta_1} && \text{(B.11)}\\
x_1 &= \pi - \omega_2 - \arccos\frac{\eta_2+\eta_3}{\zeta_2} && \text{(B.12)}
\end{aligned}$$

One can attempt to extract the actual equation of the spike curve by substitution, but the resulting form is too complicated. Furthermore, we can prove desired properties of the curve using just the parametric form. In order to prove these properties though, we need to find the equations of the slope of the curve.

We are interested in the function $d\theta/dd = \theta'/d'$, where the symbol $'$ denotes differentiation with respect to x. From Equations (B.1)-(B.2) we get:

$$\begin{aligned} \theta' &= ({\varphi_1}' + {\varphi_2}')/2 + ({\alpha_1}' + {\alpha_2}')/2 && \text{(B.13)} \\ d' &= ({\varphi_2}' - {\varphi_1}')/2 + ({\alpha_2}' - {\alpha_1}')/2 + 1 && \text{(B.14)} \end{aligned}$$

and from Equations (B.3)-(B.10):

$$\begin{aligned} {\varphi_1}' &= -\zeta_1\eta_1\eta_3 \sin(x - \omega_1)/{\mu_1}^3 \cos\varphi_1 && \text{(B.15)} \\ {\varphi_2}' &= \zeta_2\eta_2\eta_3 \sin(x + \omega_2)/{\mu_2}^3 \cos\varphi_2 && \text{(B.16)} \\ {\alpha_1}' &= ({\zeta_1}^2 - \zeta_1\eta_3 \cos(x - \omega_1))/{\mu_1}^2 && \text{(B.17)} \\ {\alpha_2}' &= -({\zeta_2}^2 + \zeta_2\eta_3 \cos(x + \omega_2))/{\mu_2}^2 && \text{(B.18)} \\ {\mu_1}' &= \zeta_1\eta_3 \sin(x - \omega_1)/\mu_1 && \text{(B.19)} \\ {\mu_2}' &= -\zeta_2\eta_3 \sin(x + \omega_2)/\mu_2 && \text{(B.20)} \end{aligned}$$

Properties of the spike curve

In Appendix A we showed that the locus of a spike of two disks for a given value of θ is a closed fourth degree curve. This curve has several interesting properties, some of which we will need, in order to derive the properties of the spike curve. In particular:

1. Not all points of a locus curve correspond to feasible spikes. We call the set of points that correspond to feasible spikes the *valid part* of the curve.
2. The valid part of a locus curve is convex.
3. The valid part of a locus curve can have at most two intersections with a circular disk.
4. The valid parts of the locus curves of two disks for all possible values of θ define a family of convex curves such that:
    (a) No two distinct curves in the family intersect.
    (b) Curves that correspond to higher θ values lie to the inside of curves corresponding to lower θ values.

(c) A circular disk can be outside tangent to at most one of these curves.

Now consider the locus of the spike of D_1 and D_2 that passes from some position of p on D_3. Let θ be the corresponding value of the uncertainty half-angle. From property 3, there can be at most one other position of p on D_3 that corresponds to the same value of θ. If we grow the value of θ, the locus will shrink, and the two points will move closer. If we continue growing θ the two points will coincide at a location p_{max} that corresponds to the maximum value of θ. The locus corresponding to this value is outside tangent to D_3.

This is the only maximum for θ. Indeed a maximum can occur only at a point where a spike locus is outside tangent to D_3. Otherwise, we would be able to move to the interior of the locus while staying on the boundary of D_3 and from property 4(b) the new point would correspond to a higher value of θ, contradicting the assumption that the initial point was a local maximum. But from property 4(c) there is a single locus curve corresponding to two given disks, that is outside tangent to a third given disk. It remains to examine the case where the interesting part of the same locus is outside tangent to D_3 at two distinct locations. This again is impossible from property 2. So we proved the following lemma:

Lemma B.1 *As p moves counterclockwisely on the boundary D_3 from its leftmost to its rightmost location, θ initially grows, reaches a global maximum, and then shrinks again.*

We can now use this result to examine the behavior of d, as p moves (see Figure B.4). Let p be located to the left of p_{max}. If the slope of the right ray is s_2, then $s_2 = d + \theta + \pi$. If p makes a small counterclockwise move, so that it remains to the left of p_{max}, the right ray rotates clockwisely by ds_2, θ grows by $d\theta$ and d changes by dd. In the new location we have: $s_2 - ds_2 = d + dd + \theta + d\theta + \pi$, where $ds_2 > 0$ and $d\theta > 0$. After simplifying we get: $dd = -ds_2 - d\theta < 0$. With similar reasoning for the left ray we can prove that $dd < 0$ when p lies to the right of p_{max}, too. And of course at p_{max}, where $d\theta = 0$, $dd = -ds_1 = -ds_2 < 0$. Note, that both rays always move clockwisely, because of the restrictions we imposed on the motion of p on D_3. Thus we proved that:

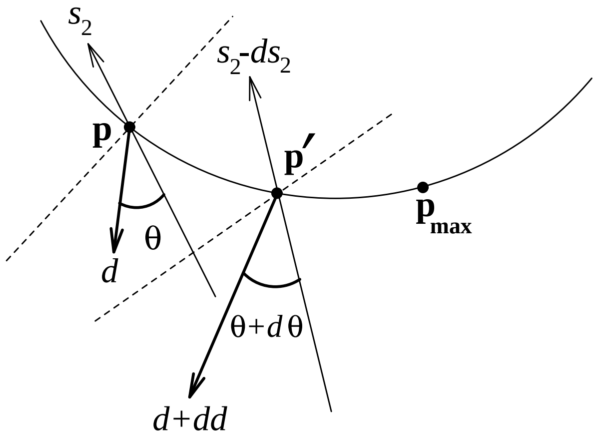


Figure B.4: Variation of the commanded velocity orientation.

Lemma B.2 *As p moves counterclockwisely on D_3 from its leftmost to its rightmost location, d moves clockwisely.*

Since d cannot assume all values in $[0, 2\pi]$ with the spike remaining on the boundary of D_3, the previous lemmas lead to this corollary:

Corollary B.3 *The spike curve is a continuous function $\theta = f_{spike}(d)$. This function grows monotonically up to some point (d_{max}, θ_{max}), and then shrinks monotonically.*

The continuity of the spike curve is intuitively obvious, so we omit the proof. We will now prove that the slope of the spike curve is always less than or equal to 1 in absolute value. We need to show that $|\theta'/d'| \leq 1$. When p moves counterclockwisely, it is easy to see that x grows. Therefore, we can apply the above lemmas to θ' and d': From Lemma B.1 we get that to the left of p_{max} θ' is positive, and to the right of it it is negative. From Lemma B.2 we get that d' is always negative. Thus, we can distinguish two cases:

(a) Right of p_{max}: Prove that $d' - \theta' \leq 0$

Using Equations (B.13) and (B.14) the above becomes: ${\alpha_1}' + {\varphi_1}' \geq 1$. Substituting from Equations (B.17) and (B.15) and simplifying yields:

$$\frac{\zeta_1 \cos(x - \omega_1) - \eta_3}{\eta_1} \geq \frac{\zeta_1 \sin(x - \omega_1)}{({\mu_1}^2 - {\eta_1}^2)^{1/2}}$$

Now notice that $x - \omega_1$ is the angle between the lines connecting c_3 with c_1 and c_3 with p. Also notice that the value of this angle, lies between the values corresponding to the leftmost position of p and the rightmost position of p. The rightmost position of p is found by drawing the common interior tangent of D_1 and D_3, and the value of the corresponding angle is $arccos((\eta_1 + \eta_3)/\zeta_1)$. The leftmost position of p is found by drawing the interior common tangent of D_2 and D_3, and always lies to the right of the point that corresponds to the other interior common tangent of D_1 and D_3. This is true because the right ray cannot go beyond the position where it becomes the right outside common tangent of D_1 and D_2. Thus the value of the angle we study is always bigger than $-arccos((\eta_1 + \eta_3)/\zeta_1)$. Consequently, the cosine of the angle is always greater than $(\eta_1 + \eta_3)/\zeta_1$. From this, it follows that:

$$\frac{\zeta_1 \cos(x - \omega_1) - \eta_3}{\eta_1} \geq 1$$

The equality holds only at the rightmost position of p. [1]

In order to complete the proof, we will show that $\zeta_1 \sin(x - \omega_1)/({\mu_1}^2 - {\eta_1}^2)^{1/2} \leq 1$. After substituting μ_1, squaring and simplifying, the above is reduced to showing that $(\zeta_1 \cos(x - \omega_1) - \eta_3)^2 \geq {\eta_1}^2$, which follows directly from the inequality we proved above. Once again the equality holds only at the rightmost position of p.

(b) Left of p_{max}: Prove that $d' + \theta' \leq 0$

This translates to ${\alpha_2}' + {\varphi_2}' \leq -1$, which can be proved in a similar fashion as above. In this case, the equality occurs at the leftmost position of p. So our proof is complete. We have:

Lemma B.4 *The slope of a spike curve never exceeds 1 in absolute value. It is equal to -1 when p is at its leftmost position, and to 1 when p is at its rightmost position.*

So, a spike curve emerges from the L-Touch critical curve that corresponds to D_1 and D_3, and ends by blending into the L-Exit critical curve that corresponds to D_2 and D_3 (see Figure B.5). In both cases the transition is smooth, because the curves have the same slope (1 and -1 respectively). We can now state the major property of a spike curve:

---

[1] And at the leftmost in the degenerate case where all three disks share a common tangent.

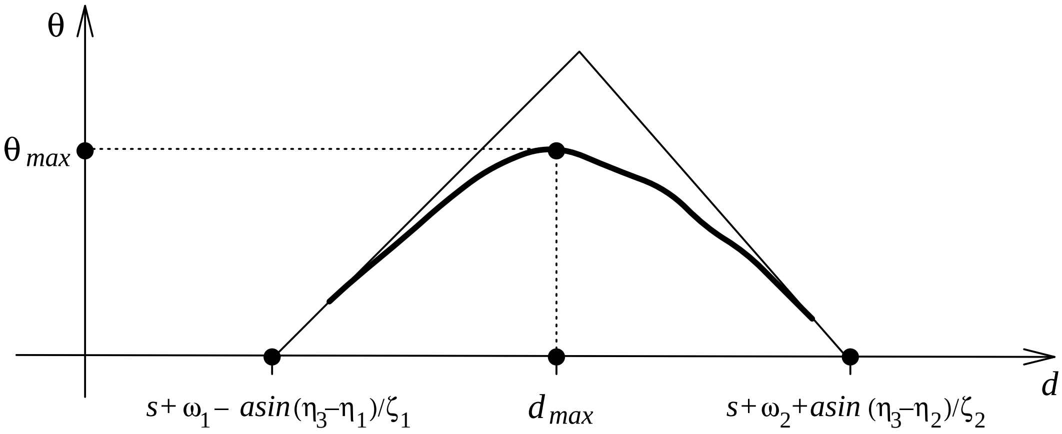


Figure B.5: Spike curve

Theorem B.5 *A spike curve has at most one intersection with any straight line of slope 1 or* -1.

This follows directly from Lemma B.4 and the fact that the spike function is continuous.

Bibliography

[1] R.C. Arkin. Towards the unification of navigational planning and reactive control. In *Working Notes of the AAAI Spring Symposium on Robot Navigation*, pages 1–5, 1989.

[2] V.I. Arnold. *Mathematical Methods of Classical Mechanics*. Springer, New York, NY, 1978.

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

[4] R. Bajcsy, E. Krotkov, and M. Mintz. Models of errors and mistakes in machine perception. Technical Report MS-CIS-86-26, GRASP LAB 64, Department of Information and Computer Science, University of Pennsylvania, Philadelphia, PA, 1986.

[5] J. Barraquand and J.C. Latombe. Robot motion planning: A distributed representation approach. Technical Report STAN-CS-89-1257, Department of Computer Science, Stanford University, Stanford, CA, 1989.

[6] R. Bolles and H. Baker. Epipolar-plane image analysis: A technique for analyzing motion sequences. In *Proceedings of the Third Workshop on Computer Vision: Representation and Control*, pages 168–178, October 1985.

[7] R.I. Brafman, J.C. Latombe, and Y. Shoham. Towards knowledge-level analysis of motion planning. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI)*, Washington, DC, July 1993.

[8] A.J. Briggs. An efficient algorithm for one-step planar compliant motion planning with uncertainty. In *Proceedings of the 5th Annual Symposium on Computational Geometry*, pages 187–196, Saarbrücken, Germany, 1989.

[9] R.A. Brooks. Symbolic error analysis and robot planning. *International Journal of Robotics Research*, 1(4):29–68, 1982.

[10] R.A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.

[11] R.A. Brooks and T. Lozano-Pérez. A subdivision algorithm in configuration space for findpath with rotation. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 799–806, Karlsruhe, Germany, 1983.

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

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

[14] J.F. Canny. On computability of fine motion plans. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 177–182, Scottsdale, AZ, 1989.

[15] J.F. Canny and J.H. Reif. New lower bound techniques for robot motion planning problems. In *Proceedings of the 28th IEEE Symposium on Foundations of Computer Science*, pages 49–60, Los Angeles, CA, 1987.

[16] D. Chapman and P.E. Agre. Abstract reasoning as emergent from concrete activity. In M.P. Georgeff and A.L. Lansky, editors, *Reasoning about Actions and Plans.* Morgan Kaufmann, Los Altos, CA, 1991.

[17] B. Chazelle, H. Edelsbrunner, L. Guibas, M. Sharir, and J. Snoeyink. Computing a face in an arrangement of line segments. *SIAM J. Comput.*, 22, 1993.

[18] A. Christiansen, M. Mason, and T.M. Mitchell. Learning reliable manipulation strategies without initial physical models. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1990.

[19] J.L. Crowley. World modeling and position estimation for a mobile robot using ultrasonic ranging. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 674–680, Scottsdale, AZ, 1989.

[20] G.A. Dakin and R.J. Popplestone. Augmenting a nominal assembly motion plan with a compliant behavior. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI)*, pages 653–658, Anaheim, CA, July 1991.

[21] E. Davis. *Representing and Acquiring Geographic Knowledge.* Courant Institute of Mathematical Sciences, New York University, Morgan Kaufmann, 1986.

[22] R.S. Desai. *On Fine Motion in Mechanical Assembly in Presence of Uncertainty.* Ph.D. Dissertation, Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI, 1988.

[23] B.R. Donald. *Error Detection and Recovery for Robot Motion Planning with Uncertainty.* Ph.D. dissertation, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, 1987.

[24] B.R. Donald. A geometric approach to error detection and recovery for robot motion planning with uncertainty. *Artificial Intelligence Journal*, 37(1-3):223–271, 1988.

[25] B.R. Donald. The complexity of planar compliant motion planning under uncertainty. *Algorithmica*, 5:353–382, 1990.

[26] B.R. Donald and J. Jennings. Planning using perceptual equivalence classes. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 190–197, Sacramento, CA, 1991.

[27] B.R. Donald and J. Jennings. Constructive recognizability for task-directed robot programming. *Journal of Robotics and Autonomous Systems*, 9:41–74, 1992.

[28] M. Drummond. Situated control rules. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 103–113, Los Altos, CA, 1989. Morgan Kaufmann.

[29] B. Dufay and J.C. Latombe. An approach to automatic robot programming based on inductive learning. *International Journal of Robotics Research*, 3(4):3–20, 1984.

[30] H. Durrant-Whyte. Concerning uncertain geometry in robotics. *Artificial Intelligence Journal*, 37, 1986.

[31] H. Edelsbrunner, L. J. Guibas, and M. Sharir. The complexity and construction of many faces in arrangements of lines and of segments. *Discrete Comput. Geom.*, 5:161–196, 1990.

[32] M. Erdmann. On motion planning with uncertainty. Technical Report AI-TR 810, AI Laboratory, MIT, Cambridge, MA, 1984.

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

[34] M. Erdmann. *On Probabilistic Strategies for Robot Tasks.* Ph.D. Dissertation, AI Laboratory, MIT, Cambridge, MA, 1990.

[35] M. Erdmann. Randomization in robot tasks. *International Journal of Robotics Research*, 11(5):399–436, 1992.

[36] M. Erdmann. Towards task-level planning: Action-based sensor design. Technical Report CMU-CS-92-116, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, February 1992.

[37] M. Erdmann and M.T. Mason. An exploration of sensorless manipulation. In *Proceedings of the IEEE International Conference of Robotics and Automation*, pages 1569–1574, San Francisco, CA, 1986.

[38] B. Faverjon and P. Tournassoud. A local based approach for path planning of manipulators with a high number of degrees of freedom. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1152–1159, Raleigh, NC, 1987.

[39] R.J. Firby. An investigation into reactive planning in complex domains. In *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI)*, pages 202–206, Seattle, WA, July 1987.

[40] A. Fox and S. Hutchinson. Exploiting visual constraints in the synthesis of uncertainty-tolerant motion plans. Technical Report UIUC-BI-AI-RCV-92-05, The University of Illinois at Urbana-Champaign, October 1992.

[41] J. Friedman. *Computational Aspects of Compliant Motion Planning.* Ph.D. Dissertation, Department of Computer Science, Stanford University, Stanford, CA, 1991.

[42] J. Friedman, J. Hershberger, and J. Snoeyink. Compliant motion in a simple polygon. In *Proceedings of the 5th ACM Symposium on Computational Geometry*, Saarbrücken, Germany, 1989.

[43] K.Y. Goldberg. *Stochastic Plans for Robotic Manipulation.* Ph.D. Dissertation, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1990.

[44] S.N. Gottschlich and A.C. Kak. Dealing with uncertainty in cad-based assembly motion planning. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI)*, pages 646–652, Anaheim, CA, July 1991.

[45] L. Gouzènes. Strategies for solving collision-free trajectories problems for mobile and manipulator robots. *International Journal of Robotics Research*, 3(4):51–65, 1984.

[46] L.J. Guibas and J. Stolfi. Ruler, compass and computer: The design and analysis of geometric algorithms. Technical Report 37, Digital Equipment Corp., Systems Research Center, Palo Alto, CA, 1989.

[47] M. Hebert and T. Kanade. First results on outdoor scene analysis using range sensors. In *Proceedings of the Image Understanding Workshop*, pages 224–231, Miami Beach, FL, December 1985.

[48] J.E. Hopcroft and G. Wilfong. Motion of objects in contact. *International Journal of Robotics Research*, 4(4):32–46, 1986.

[49] S. Hutchinson. Exploiting visual constraints in robot motion planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1722–1727, Sacramento, CA, 1991.

[50] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.

[51] A. Koutsou. *Planning Motion in Contact to Achieve Parts Mating.* Ph.D. Dissertation, Department of Computer Science, University of Edinburgh, UK, 1986.

[52] J.C. Latombe. Motion planning with uncertainty: The preimage backchaining approach. Technical Report STAN-CS-88-1196, Department of Computer Science, Stanford University, Stanford, CA, 1988.

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

[54] J.C. Latombe, A. Lazanas, and S. Shekhar. Motion planning with uncertainty: Practical computation of non-maximal preimages. In *Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems*, Tsukuba, Japan, 1989.

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

[56] C. Laugier. Planning fine motion strategies by reasoning in the contact space. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 653–659, Scottsdale, AZ, 1989.

[57] C. Laugier and F. Germain. An adaptative collision-free trajectory planner. In *Proceedings of the International Conference on Advanced Robotics*, Tokyo, Japan, 1985.

[58] C. Laugier and P. Théveneau. Planning sensor-based motions for part-mating using geometric reasoning techniques. In *Proceedings of the 8th European Conference on Artificial Intelligence*, Brighton, UK, 1986.

[59] A. Lazanas and J.C. Latombe. Landmark-based robot navigation. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI)*, pages 697–702, San Jose, CA, July 1992.

[60] A. Lazanas and J.C. Latombe. Landmark-based robot navigation. Technical Report STAN-CS-92-1428, Department of Computer Science, Stanford University, Stanford, CA, 1992. To appear in Algorithmica.

[61] A. Lazanas and J.C. Latombe. Landmark-based robot motion planning. In C. Laugier, editor, *Geometric Reasoning: From Perception To Action (Lecture Notes in Computer Science)*. Springer Verlag, 1993.

[62] A. Lazanas and J.C. Latombe. Motion planning with uncertainty: A landmark approach. *Artificial Intelligence Journal; Special Issue on Planning and Scheduling*, 1993. To appear.

[63] J.J. Leonard and H.F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, RA-7(3):376–382, 1991.

[64] T.S. Levitt, D.T. Lawton, D.M. Chelberg, and P.C. Nelson. Qualitative navigation. In *Proceedings of the Image Understanding Workshop*, pages 447–465, Los Angeles, CA, 1987.

[65] L.I. Liebermann and M.A. Wesley. Autopass: An automatic programming system for computer controlled mechanical assembly. *The IBM Journal of Research and Development*, 21(4):321–333, 1977.

[66] T. Lozano-Pérez. The design of a mechanical assembly system. Technical Report AI-TR 397, Artificial Intelligence Laboratory, MIT, Cambridge, MA, 1976.

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

[68] T. Lozano-Pérez. A simple motion-planning algorithm for general robot manipulators. *IEEE Transactions on Robotics and Automation*, RA-3(3):224–238, 1987.

[69] T. Lozano-Pérez, M.T. Mason, and R.H. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1):3–24, 1984.

[70] S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. Research report, IBM T.J. Watson Center, Yorktown Heights, NY, 1990.

[71] M.T. Mason. Compliance and force control for computer controlled manipulators. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(6):418–432, 1981.

[72] M.T. Mason. Automatic planning of fine motions: Correctness and completeness. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 492–503, Atlanta, GA, 1984.

[73] M.J. Mataric. Behavior-based systems: Key properties and implications. In *Proceedings of the IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems*, pages 46–54, Nice, France, 1992.

[74] T.M Mitchell. An approach to concept learning. Technical Report STAN-CS-78-711, Department of Computer Science, Stanford University, Stanford, CA, 1978.

[75] H.P. Moravec. *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover.* Ph.D. Dissertation, Stanford University, Stanford, CA, 1976.

[76] H.P. Moravec and A.E. Elfes. High resolution maps from wide angle sonar. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 116–121, St. Louis, MO, 1985.

[77] B.K. Natarajan. The complexity of fine motion planning. *International Journal of Robotics Research*, 7(2):36–42, 1988.

[78] J. Nievergelt and F.P. Preparata. Plane-sweep algorithms for intersecting geometric figures. *Communications of the ACM*, 25(10):739–747, October 1982.

[79] N.J. Nilsson. *Principles of Artificial Intelligence.* Morgan Kaufman, Los Altos, CA, 1980.

[80] A. Pentland. A new sense for depth of field. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 988–994, Los Angeles, CA, 1985.

[81] J. Pertin-Troccaz and P. Puget. Dealing with uncertainty in robot planning using program proving techniques. In *Proceedings of the 4th International Symposium on Robotics Research*, pages 455–466, Santa-Cruz, CA, 1987.

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

[83] M.H. Raibert and J.J. Craig. Hybrid position/force control of manipulators. *Journal of Dynamic Systems, Measurement and Control*, 102:3–18, 1981.

[84] V.T. Rajan, R. Burridge, and J.T. Schwartz. Dynamics of rigid body in frictional contact with rigid walls. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 671–677, Raleigh, NC, 1987.

[85] J.H. Reif. Complexity of the mover's problem and generalizations. In *Proceedings of the 20th Symposium on the Foundations of Computer Science*, pages 421–427, 1979.

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

[87] J.T. Schwarz and M. Sharir. On the piano movers' problem ii: General techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4:298–351, 1983.

[88] J.T. Schwarz and M. Sharir. A survey of motion planning and related geometric algorithms. *Artificial Intelligence Journal*, 37(1-3):157–169, 1988.

[89] M. Sharir. Efficient algorithms for planning purely translational collision-free motion in two and three dimensions. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1326–1331, Raleigh, NC, 1987.

[90] S. Shekhar and O. Khatib. Force strategies in real-time fine motion assembly. In *Proceedings of the ASME Winter Annual Meeting*, Boston, MA, 1987.

[91] H. Takeda and J.C. Latombe. Sensory uncertainty field for mobile robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2465–2472, Nice, France, 1992.

[92] R.H. Taylor. *Synthesis of Manipulator Control Programs from Task-Level Specifications.* Ph.D. Dissertation, Department of Computer Science, Stanford University, Stanford, CA, 1976.

[93] A. Timcenko and P. Allen. Modeling uncertainties in robot motions. In *Working Notes of the AAAI Fall Symposium on Applications of Artificial Intelligence to Real-World Autonomous Mobile Robots*, pages 137–145, Cambridge, MA, 1992.

[94] J. Valade. Automatic generation of trajectories for assembly tasks. In *Proceedings of the 6th European Conference on Artificial Intelligence*, Pisa, Italy, 1984.

[95] R. Waldinger. Achieving several goals simultaneously. In E. Elcock and D. Michie, editors, *Machine Intelligence 8.* Ellis Horwood, Chichester, UK, 1975.

[96] D. Whitney. Force feedback control of manipulator fine motions. *ASME Journal of Dynamic Systems, Measurement, and Control*, 99:91–97, 1977.

[97] Z. Zhang and O. Faugeras. A 3D world model builder with a mobile robot. *International Journal of Robotics Research*, 11(4):269–285, 1992.