EFFICIENT GEOMETRIC ALGORITHMS FOR ROBOT SENSING AND CONTROL

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Amy Judith Briggs

January 1995

EFFICIENT GEOMETRIC ALGORITHMS
FOR ROBOT SENSING AND CONTROL

Amy Judith Briggs, Ph.D.
Cornell University 1995

This thesis addresses the problem of automatically generating solutions to robotics tasks that are specified at a high level. In particular, we consider the problems of robot motion planning and the planning of sensor placements.

These problems are made difficult by a number of inherent factors. Foremost among these are *uncertainty* and *geometric complexity*. Uncertainty arises from the fact that the actions of robots are subject to error. Geometric complexity reflects the fact that real-world task environments are often complex. If we want robot strategies that are both practical and robust, we must develop algorithms that successfully deal with uncertainty and complex geometry.

Much of the previous work in the area of task-level planning for robots fails to address at least one of these issues. Many theoretical approaches are algorithmically sophisticated, but do not handle uncertainty, and may be unimplementable in practice. On the other hand, real robot systems often employ simplistic strategies that do not take into account complex geometric interactions. This thesis seeks to bridge the gap between these two extremes. We present efficient planning algorithms for motion and sensing that are both practical and algorithmically sophisticated.

Our motion planning algorithm computes one-step motion strategies that guarantee reaching a specified goal in the plane. To deal with uncertainty in robot control, we employ a control model that allows the robot to slide along obstacle surfaces, or *comply* with the environment. Our analysis of this algorithm yields a precise characterization of the complexity of one-step compliant motion planning with uncertainty.

Sensors are needed within autonomous systems to provide execution-time feedback. In this thesis we develop a framework for planning sensing strategies in a principled way. In particular, we present algorithms for computing the set of placements from which a sensor can monitor a region within a task environment. This work has many applications in the areas of assembly planning, cooperating robots, and robot surveillance. We have demonstrated the practicality of our approach by building a system of robot surveillance with mobile robots employing our strategies for sensor planning.

Biographical Sketch

Amy Judith Briggs was born on October 26, 1964 in Manchester, New Hampshire. She studied computer science at Dartmouth College, where she received a Bachelor of Arts degree in 1986. After graduation she moved to the Silicon Valley, where she spent a year working as a software engineer at Hewlett-Packard. In 1987 she left the California sunshine to come back east, and enrolled in the Computer Science graduate program at Cornell University. There, she received a Master of Science degree in 1990 and met the love of her life, Daniel Scharstein, whom she married on April 24, 1992. She received her Ph.D. in January 1995.

To my parents

Acknowledgements

I would first like to thank my family for all their support over the years, and for giving me the many opportunities that have made this achievement possible. This thesis is dedicated to my parents with love and gratitude.

The Cornell Computer Science Department has been a wonderful place to learn and work. I am indebted to many people here who have taught and inspired me. I am most grateful to my advisor Bruce Donald for his guidance, support, and encouragement throughout my time here. Bruce's ideas, wisdom, and good judgement permeate this thesis. I also thank my other committee members, Dan Huttenlocher, Dexter Kozen, Jerry Marsden, and proxy member Moss Sweedler, who have all assisted me in many different ways throughout my career as a graduate student. Thanks to Bruce, Dexter, and Dan for carefully reading drafts of the thesis and giving me valuable comments and suggestions.

I am grateful to the AT&T Foundation for their generous support in the form of a Graduate Fellowship.

Many thanks to my friends Klara Kedem and Paul Chew who have discussed much of the material in this thesis with me. They have always been very generous with their time and ideas.

The work described in Chapter 2 benefitted from discussions with John Canny, Joe Mitchell, Randy Brost, and Samir Khuller. The work described in Chapter 3 grew out of discussions at the MSI Workshop on Computational Robotics held at Cornell in June 1991. Thanks to Bruce Donald, Mike Erdmann, Tomás Lozano-Pérez, and Matt Mason for their inspiration and the ideas they shared at this meeting.

I am very grateful to the many people in the Cornell Robotics and Vision Laboratory who contributed to the development and success of the work I describe in Chapter 5. Bruce Donald provided the inspiration and guidance which started the project in the first place, as well as all the tools to make it possible. Mike Leventon wrote an early version of the code which was ultimately loaded onto the robots and used to interface with them and the recognition system. His expertise and help in starting the project were invaluable. Daniel Scharstein helped me with the vision system and also spent many long hours helping to get everything working just right. Thanks to Russell Brown and Jim Jennings who built the robots, and who also patiently answered all my questions about running the robots. Thanks to William Rucklidge who wrote the vision recognition system, and to Jonathan Rees who wrote the mobot programming environment, in addition to the geometric modeling system I used.

I had the good fortune to spend a summer at the Stanford University Robotics Lab while working on the research described in this thesis. Thanks to my host Jean-Claude Latombe, and also to Dan Halperin, Leo Guibas, and Tsai-Yen Li for helpful discussions.

Finally, I wish to express my deepest gratitude to my husband, Daniel, for all the help and support he has given me, and especially for all the happiness he has brought to my life.

This thesis describes research done in the Robotics and Vision Laboratory at Cornell University. Support for our robotics research is provided in part by the National Science Foundation under grants No. IRI-8802390, IRI-9000532, IRI-9201699, and by a Presidential Young Investigator award to Bruce Donald, and in part by the Air Force Office of Sponsored Research, the Mathematical Sciences Institute, Intel Corporation, and AT&T Bell laboratories.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis addresses the problem of automatically generating solutions to robotics tasks that are specified at a high level. Algorithms are given for both robot motion planning and the planning of sensor placements. These algorithms give us answers to questions of the form:

1. Given a robot at some location in the world, can the robot negotiate around obstacles in order to reach a desired goal? Can such a motion plan be computed efficiently and robustly?

2. Where should a sensor be placed so that a specified region within a task environment can be monitored? How can we configure sensors to determine the success or failure of a motion plan?

These tasks are made difficult by a number of inherent factors. Foremost among these are *uncertainty* and *geometric complexity*.

Uncertainty in robot planning arises from errors in robot control and sensing, and in the geometric models of task environments. Since real robots must successfully execute tasks in the presence of uncertainty, we must take uncertainty into account when planning robot strategies. Geometric complexity reflects the fact that the environments in which robots operate typically include many objects and therefore have complex geometric descriptions.

Much of the previous work in the area of task-level planning for robots fails to address at least one of these issues. Many theoretical approaches are algorithmically sophisticated, but do not handle uncertainty, and may be unimplementable in practice. On the other hand, real robot systems often employ simplistic strategies that do not take into account complex geometric interactions. This thesis seeks to bridge the gap between these two extremes. We present efficient planning algorithms for motion and sensing that are both practical and algorithmically sophisticated.

To understand some of the difficulties that uncertainty introduces, consider the simple mechanical assembly problem of inserting a peg in a hole, as illustrated

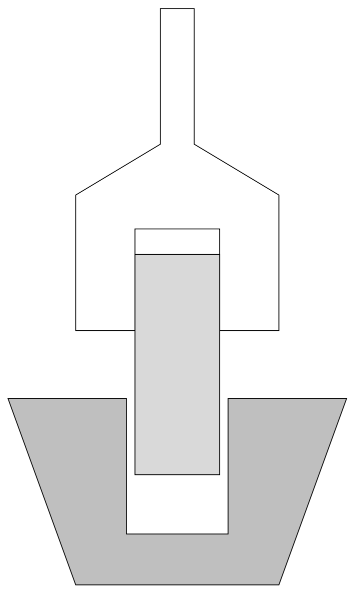

Figure 1.1: A peg-in-hole task. The lightly shaded object represents a peg, which is grasped by a mechanical gripper and inserted into a hole.

in Figure 1.1. Suppose we had a robotic assembly system that was capable of gripping the peg and performing the task. How should a planner program the robot to perform the insertion? What if the tolerance between the peg and hole were tight? We want our generated plans to anticipate potential problems, for example, missing the hole or jamming, as depicted in Figure 1.2.

Allowing the peg to slide along obstacle surfaces is one way to overcome these difficulties. We call this type of motion *compliant* motion, because the robot or manipulated part *complies* with the environment. See Figure 1.3 for an illustration of a compliant motion. We will examine a special case of compliant motion planning with uncertainty in Chapter 2. In particular, we give an algorithm for computing the set of all motion directions that guarantee reaching a specified goal region from a specified starting region, while considering uncertainty and allowing compliance on obstacle surfaces. This work builds on the foundational research in fine-motion planning by Lozano-Pérez, Mason and Taylor [LMT84], Erdmann [Erd84,Erd86], and Donald [Don89,Don90].

Given that we can *plan* motion strategies, how do we determine the success or failure of a strategy at execution time? Typically, sensors are used within autonomous systems to provide execution time feedback. In this thesis we develop a framework for planning sensing strategies in a principled way. Our algorithms for sensor planning can be used in conjunction with a motion planner (such as the one we describe in Chapter 2) to detect when a goal has been reached.

Consider the problem of monitoring a particular region (for example, a goal region) within a task environment. A cluttered environment with many obstacles

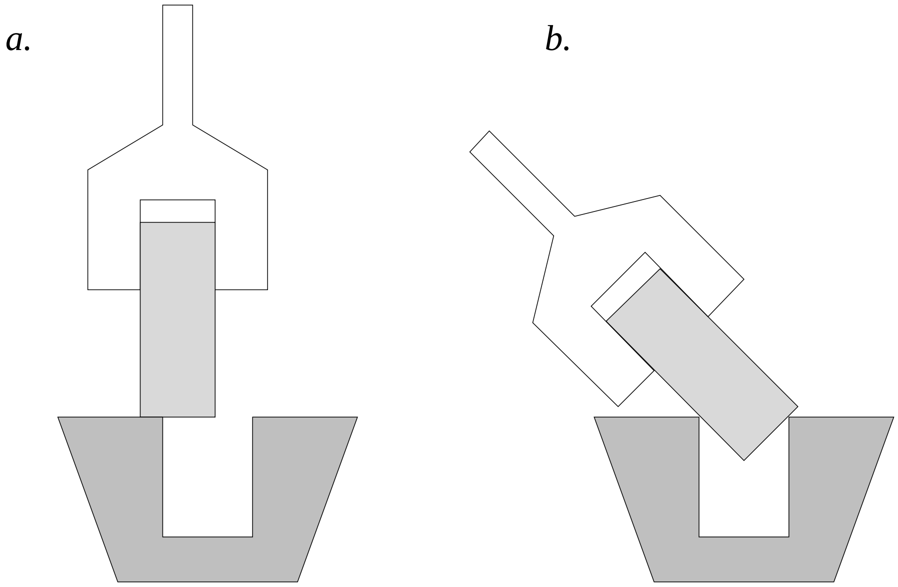

Figure 1.2: The peg misses the hole in a and jams in b.

makes it difficult to find a sensor placement that affords an unoccluded view of the region of interest. So we must consider geometric complexity when planning sensor placements, and deal with it in an efficient way. In this thesis we develop algorithms for finding sensor placements that afford a complete or partial view of a region within a cluttered environment in the plane. These algorithms have been implemented and used in a robot surveillance system. The algorithms have been extended to handle uncertainty in sensor placement and aim. We also describe algorithms for computing the set of all placements from which an object moving within a region can be monitored.

In the robot surveillance system we have developed, one mobile robot, TOMMY, has the task of monitoring a particular region within a room. A position from which TOMMY can monitor this region is determined using the algorithms we present in Chapter 3. In particular, TOMMY monitors a doorway of our robotics lab, ignoring all intruders until a another mobile robot, LILY, enters the doorway. TOMMY announces when LILY appears.

Figure 1.4 shows a picture of the mobile robots TOMMY and LILY. Figure 1.5 shows the map of the robotics lab that we use with the system. TOMMY's task is to monitor the doorway, which is marked in the Figure with "G". The dark gray regions are obstacles representing real objects in the room — chairs, desks, couches, bookshelves, *etc.* Using this map, the algorithms that we present in Chapter 3 give us the exact placements from where the doorway can be monitored. For example, the lightly shaded region in Figure 1.6 is the exact set of placements from where the doorway can be entirely seen with a sensing device such as a CCD camera. Figure 1.7 shows a picture of LILY in the doorway; the picture was taken with the

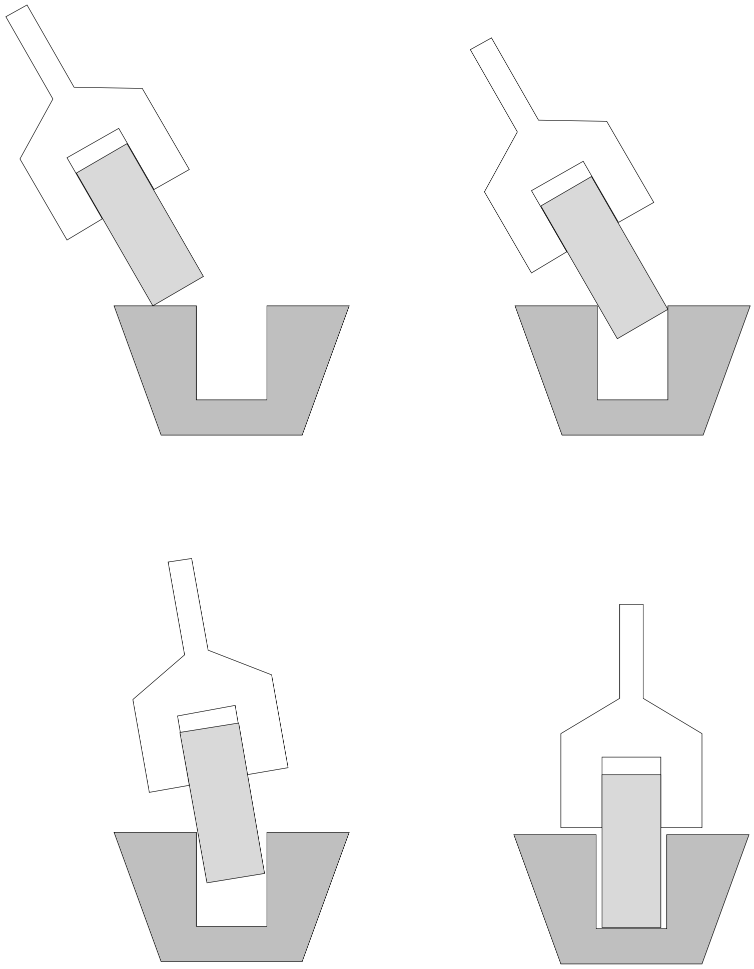

Figure 1.3: The peg complies with the environment and ends in the hole.

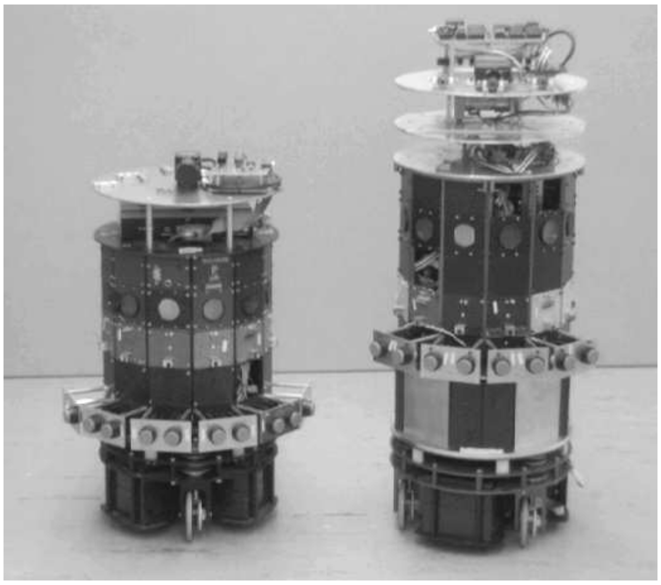

Figure 1.4: The mobile robots TOMMY and LILY. TOMMY is on the left, LILY is on the right.

camera mounted on top of TOMMY after TOMMY had moved to a configuration computed by the system.

Our primary motivations in this thesis are to deal with uncertainty and to plan sensing actions in a principled way. In our solutions to these problems, we use a combination of techniques from robotics, active sensing, and computational geometry. The next section reviews some of the fundamental concepts and definitions from these areas that we rely on throughout the thesis. Section 1.2 introduces some of the techniques we employ for dealing with uncertainty and complexity. In Section 1.3 we survey related research. We close the chapter with a summary of thesis contributions and an outline of the rest of the thesis.

1.1 Some fundamental concepts and techniques

Our algorithms exploit the geometry of the task environment in order to obtain efficient and precise solutions to problems in motion planning and sensor planning. Before we begin our discussion of these algorithms, we review some of the basic building blocks on which the thesis relies.

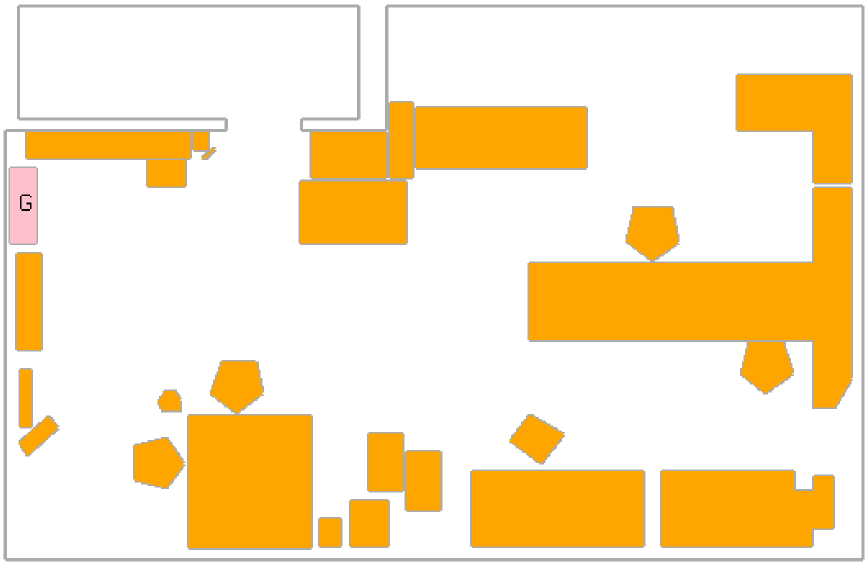

Figure 1.5: Our map of the robotics lab.

Figure 1.6: The lightly shaded region represents the set of placements from which the doorway can be completely seen.

Figure 1.7: LILY in a doorway of the robotics lab. This picture was taken with the videocamera mounted on top of the robot TOMMY, after TOMMY had moved to a computed position from which the doorway could be monitored.

1.1.1 Motion planning

Early work in motion planning solved the so-called "piano movers problem" or *findpath problem*, which is to find a collision-free path from a specified initial position to a goal position for a moving object in the presence of known obstacles. This problem is often referred to as the *classical Movers problem* [Rei79]. Motion paths were typically sought under the following assumptions:

- perfect models of the environment
- perfect control
- perfect sensing.

While our primary motivation is to address uncertainty in motion planning, we rely on the tools developed to solve the motion planning problem without uncertainty, so we review these definitions and techniques here.

A *motion plan* is a robot control strategy, usually computed before execution time. When a robot executes a motion plan, its motion traces out a *path*. When computing a motion plan, we usually have some geometric representation of the task environment. The environment consists of a starting point or region for the robot, a goal point or region that should be reached, and obstacles which the robot should avoid. Consider, for example, the environment of Figure 1.8. The shaded regions represent objects which should be avoided — they might represent a couch, a chair, and a desk that have been projected down onto the floor to form obstacle

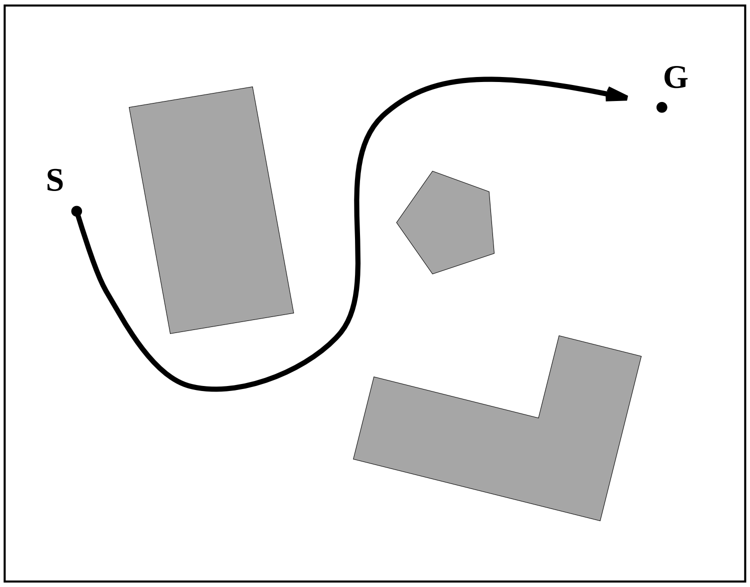

Figure 1.8: An example motion path. The curve represents the path traced out by a point robot moving from the start point S to the goal point G.

polygons. The black curve represents a motion path for a point robot that avoids the obstacles.

In designing motion planning systems, we often have different criteria for what constitutes a good path. We may, for example, require the path to be the shortest possible path. Or, we may require that the moving robot always stay maximally clear of obstacles. Or we could relax this restriction and allow the robot to slide on obstacles. There are many variations of the basic motion planning problem, even in two dimensions.

We now examine some of the simple and elegant constructs that allow us to solve some of these motion planning problems efficiently. We'll rely on these constructs throughout the thesis.

1.1.2 The visibility graph

One useful tool is the visibility graph. Given a set of polygons, the visibility graph consists of edges between vertices of the polygons. Each pair of polygon vertices that can see each other make up a *visibility edge*. The set of all such edges is called the *visibility graph*. Many computational robotics problems, for example, motion planning without uncertainty, can be solved using visibility calculations. This is because an ideal straight line motion trajectory is the path followed by light, so computing the set of reachable points reduces to computing the set of visible points. Furthermore, the visibility graph exactly characterizes the shortest paths between vertices of the polygons in an environment. If we want to find the shortest motion

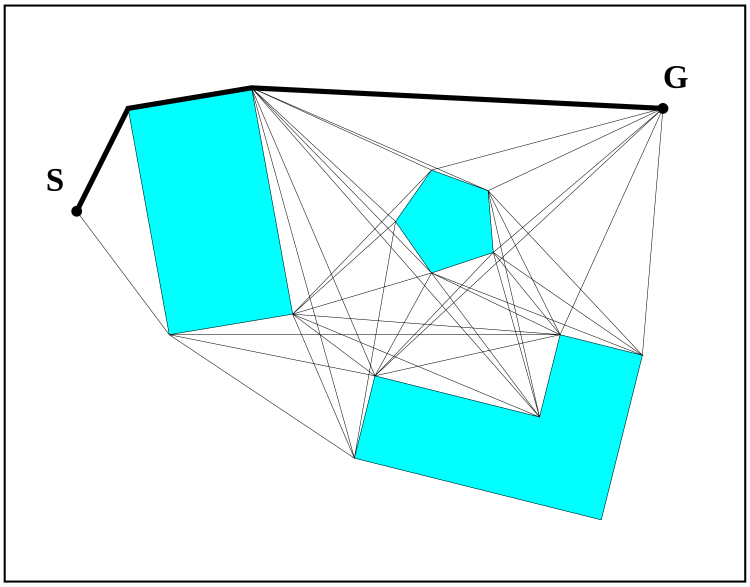

Figure 1.9: The visibility graph for the environment of Figure 1.8. The thick line represents the shortest path from S to G.

path among a set of polygons, we can construct the visibility graph and search it to find the shortest path [LW79].

Figure 1.9 shows the visibility graph for the environment of Figure 1.8. The shortest path from S to G is represented by the thick line; we see that a part of this path coincides with the boundary of one of the obstacles.

1.1.3 Configuration space

Of course, robots have geometry; they are not points. Another useful technique allows us to solve the same motion planning problem as above for a robot with a polygonal shape. See Figure 1.10 for an example. We want to find a path that will enable the robot to maneuver from S to G by translation only, *i.e.*, without rotating. The idea is that we transform the environment of obstacles and represent them in a space where we can consider the robot to be a point. The transformed space is called the *configuration space* (or *c-space* for short), and an obstacle in this space is called a *configuration space obstacle*.

As motivation for our discussion of configuration space, we note here that research on the problem of motion planning with uncertainty arose in response to the need to relax the idealizing assumptions of perfect models, perfect control, and perfect sensing. The resulting algorithms operate in c-space, and exploit the computational-geometric properties of sliding on surfaces. In order to understand the inadequacies of the above assumptions and the computational tools we employ, it is first necessary to understand the structure of c-space, and how it may be com-

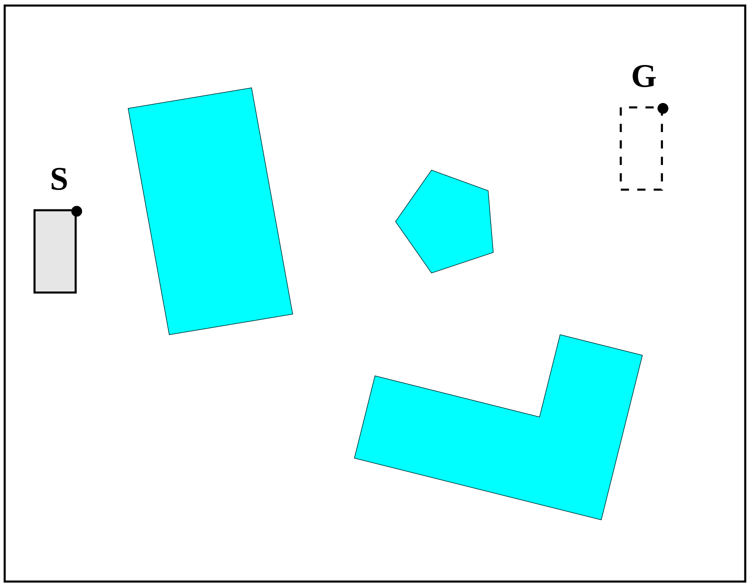

Figure 1.10: The task environment of Figure 1.8, now with a polygonal robot.

puted. To this end, we wish to give a flavor here for the kinds of issues that arise in planning paths in c-space. We examine a special case — the planar polygonal case. First we show how to describe c-space obstacles. Next we carefully define some geometric tools for their efficient computation. Finally, we hint at how to compute collision-free paths using the representations we have developed. For this simple planning problem, we find that simple geometric tools suffice. We will build on the tools presented in this section to incorporate uncertainty, compliance, and active sensing, while retaining the crisp geometric flavor and the ability of these algorithms to handle complex geometries.

We now outline the configuration space construction developed by Lozano-Pérez [Loz83]. We first choose a reference point somewhere on the robot. When we are done with the construction, we can consider the robot to be just this point. The boundary of the configuration space obstacle is called a *fence*; the fence is traced out by the reference point as we slide the robot (actually, the polygon representing the robot) around the outside of the obstacle. So the idea of the c-space construction is to build a fence around each of the obstacles, and as long as the reference point lies outside of the fences, the robot will not hit any of the obstacles. We illustrate the construction of configuration space obstacles in Figure 1.11 with a single convex obstacle and a convex robot. With slight modifications, the construction works for non-convex polygons as well.

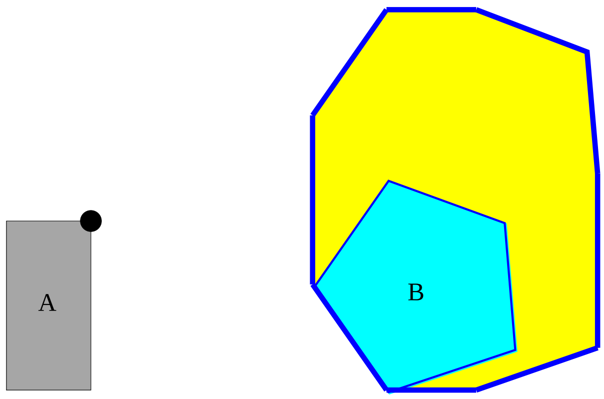

Figure 1.11: The rectangle on the left represents a robot; the pentagon represents an obstacle; the region around the pentagon is the configuration space obstacle.

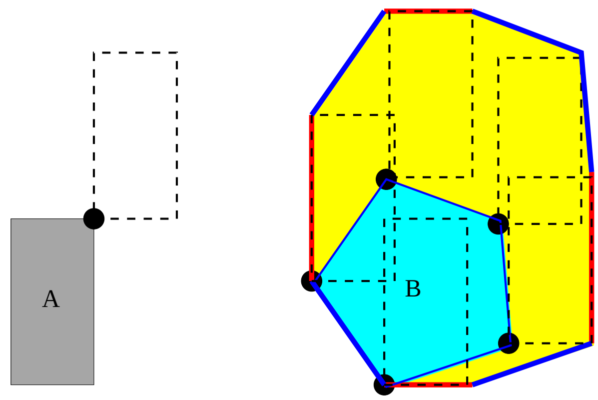

Figure 1.12: An illustration of the configuration space construction. The rotated polygon is placed at each vertex of the obstacle; the c-space obstacle is the convex hull of the result.

The construction for a single convex obstacle and a convex robot works as follows:

1. Take the polygon representing the robot and rotate it around its reference point by 180 degrees.
2. Place a copy of the rotated robot at each of the vertices of the obstacle.
3. Compute the convex hull of all the rotated polygons.

Figure 1.12 illustrates this construction.

To formalize, we make the following definitions for point sets A and B [Loz83]:

1. The *Minkowski sum* or *convolution* of A and B is defined as

$$A \oplus B = \{a + b \mid a \in A, \quad b \in B\}.$$

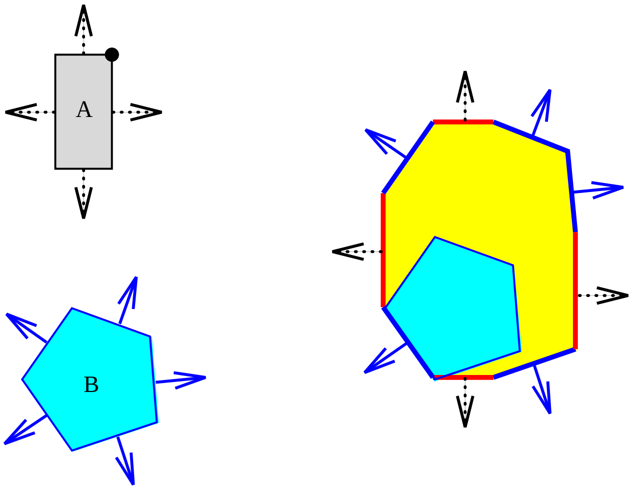

Figure 1.13: An illustration of the $O(n)$ configuration space algorithm.

2. The *reflection* of A is defined as

$$\ominus A = \{-a \mid a \in A\}.$$

3. The *configuration space obstacle* of B with respect to A is defined as

$$\begin{aligned} B \ominus A &= B \oplus (\ominus A) \\ &= \{b - a \mid b \in B, a \in A\} \end{aligned}$$

In these definitions, A and B are sets of points, and a and b are vectors representing the coordinates of points of A and B in a coordinate system with its origin at the reference point of the robot.

To compute the boundary of the c-space obstacle, or the *fence*, we note that it consists only of edges of the robot and the obstacle. Each edge of $\ominus A$ and B has an outward-facing normal. The edges of the fence can be output in order by sorting them by the angle of their normals with respect to the x-axis. But for the case of convex polygons, the normals are already sorted. So the algorithm to compute the fence in this case is simply the following:

1. Merge the edges of A and B according to the angle of their outward facing normals.
2. Concatenate the merged edges to construct the fence.

Since the algorithm essentially only merges two sorted lists, its running time is $O(n)$ where n is the total number of edges of A and B. See Figure 1.13 for an illustration.

Returning now to our motion planning example, the configuration space is shown in Figure 1.14. Note that one of the obstacles is not convex. The technique

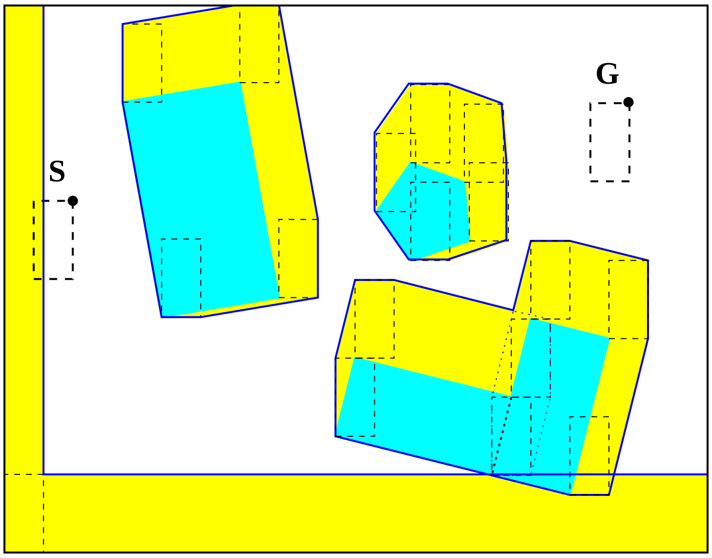

Figure 1.14: The configuration space for our motion planning example.

used in this case is to break the non-convex polygon into convex sub-polygons; then apply the algorithm above to each convex sub-polygon. Note also that in our example, the robot has two degrees of freedom: the x- and y-components of the translation, in which case the configuration space is $\mathbb{R}^2$. The configuration space construction also works for robots with more degrees of freedom. For example, for a robot that can rotate as well as translate, the configuration space becomes $\mathbb{R}^2 \times S^1$, where S^1 is the unit circle.

Now, we can build the visibility graph in this new space to represent the set of paths from S to G. Searching this graph for the shortest path gives the shortest motion path for the reference point, while the robot stays clear of obstacles. Figure 1.15 shows this path for the configuration space of Figure 1.14. We see that because of the robot's geometry, the robot can no longer move around the obstacles — the shortest collision-free path now goes between the obstacles.

Note that the shortest path contacts obstacles. Given any uncertainty in robot control, a position-controller cannot possibly execute it. If the controller instead supports compliant motion, then such paths can be executed. The input to such a compliant motion controller is not a path, but a sequence of forces to apply. Hence, if we want compliant motions, we should not synthesize shortest paths and expect a position controller to execute them accurately. Instead, we should start with the geometry of the environment and calculate forces that result in acceptable compliant motions when fed to an appropriate compliant motion controller.

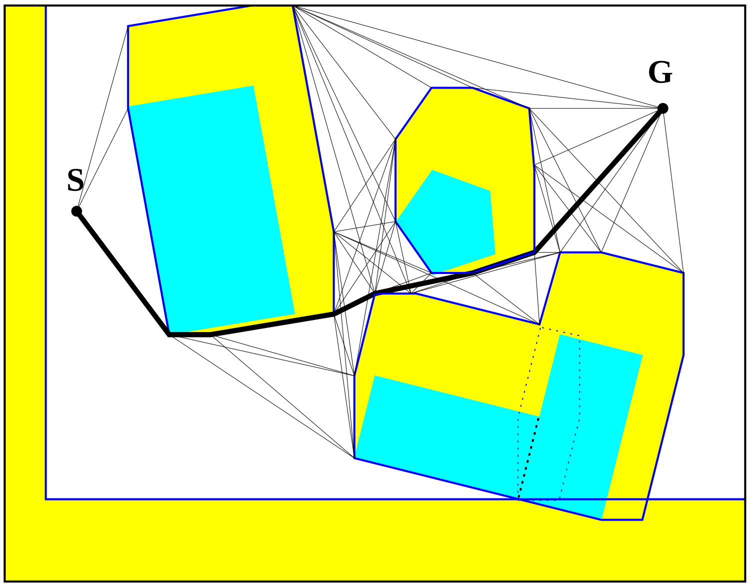

Figure 1.15: The visibility graph for the configuration space environment. The shortest collision-free motion path from S to G is drawn in bold.

1.1.4 Extensions to the basic model

The visibility graph and configuration space are two of the very basic tools used in motion planning. There are many extensions to this basic definition of motion planning to deal with the practical issues that arise in real world situations. Chief among these extensions is the consideration of *uncertainty*. Real robots are subject to errors in control and sensing, so motion planning systems must account for uncertainty if they are to be robust. We discuss some techniques for dealing with uncertainty in motion planning in Section 1.2 and in Chapter 2.

1.1.5 Art gallery theory

Research in the area of *art gallery theory* has introduced and addressed many problems pertaining to polygon visibility. The *art gallery problem* is to determine the minimum number of guards sufficient to guard the interior of a simple polygon. In our work in sensor configuration planning, we address the related question of where sensors should be placed in order to monitor a region of interest. For example, in the robot surveillance system described in Chapter 5, a sensor placement is computed from which the robot TOMMY can monitor a particular doorway of the robotics lab and announce when the robot LILY enters. Since the art gallery theory has direct relevance to our work in sensor configuration planning, we review some of the basic concepts and definitions from this theory here.

In the basic model, each guard stands in one place and can see in all directions. So the interior polygon edges that are "guarded" by a point guard are those which

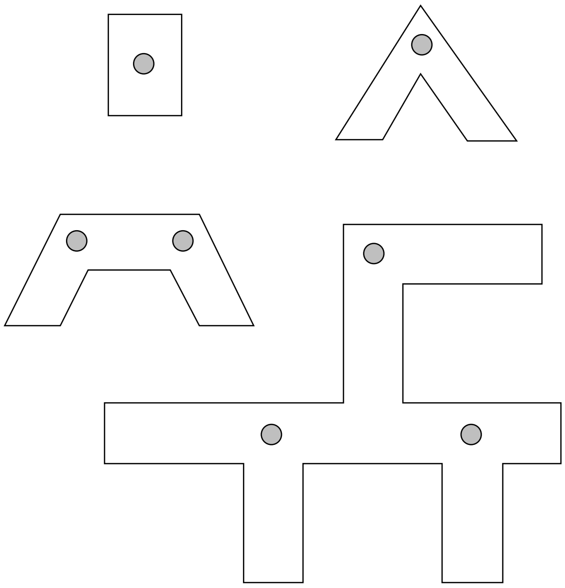

Figure 1.16: Some examples of art galleries. The gray dots represent placements of the minimal number of guards.

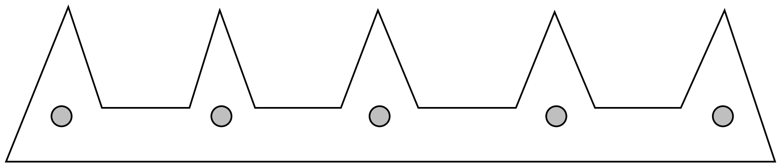

Figure 1.17: A polygon where $\lfloor n/3 \rfloor$ guards are necessary.

are not occluded. Figure 1.16 shows some examples of "art galleries" and possible placements for guards.

It has been shown that the minimum number of point guards sufficient to guard the interior of any simple polygon with n edges is $\lfloor n/3 \rfloor$. This many guards is also sometimes necessary, as demonstrated in Figure 1.17.

An interesting extension is the case of *mobile guards*, *i.e.*, guards which can patrol a line segment contained in the polygon. If s is a line segment contained in polygon P, we say that a point $x \in P$ is *covered by* s if there is a point y on s such that the line segment xy lies entirely in P. See Figure 1.18 for an illustration of a polygon covered by a single mobile guard, demonstrating the fact that mobile guards are more powerful than point guards. It has been shown that the minimum number of mobile guards sufficient to cover the interior of any simple polygon with n edges is $\lfloor n/4 \rfloor$. This many guards is also sometimes necessary, as demonstrated in Figure 1.19.

See the book by O'Rourke [O'R87] for more details and information on art gallery theory.

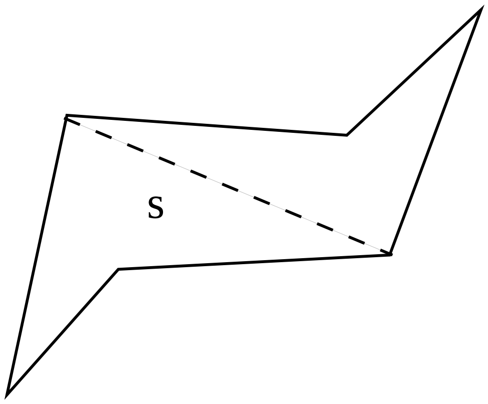

Figure 1.18: A polygon that requires one mobile guard, which moves along line segment s.

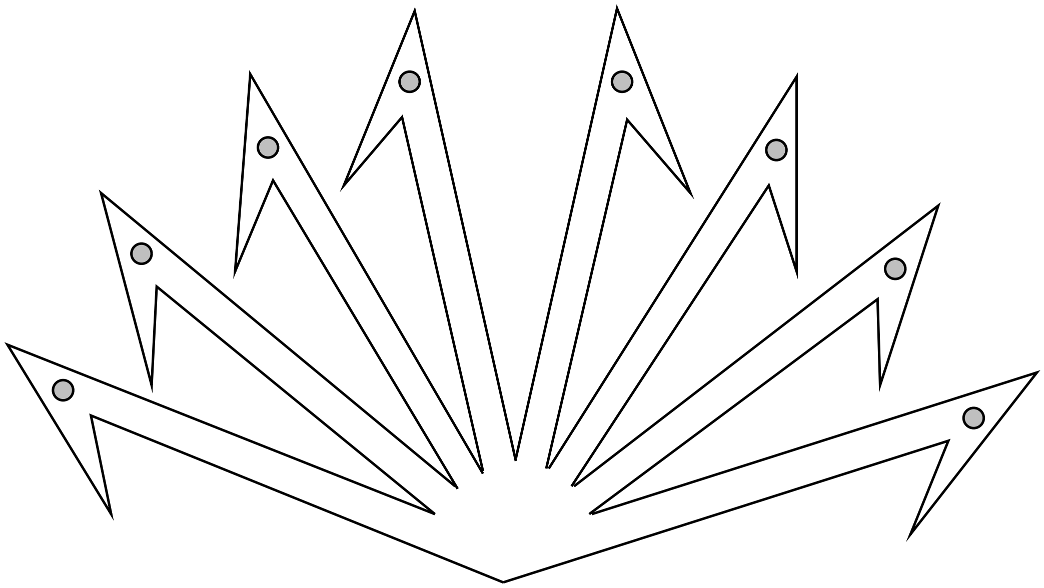

Figure 1.19: A polygon where $\lfloor n/4 \rfloor$ mobile guards are necessary. (In this example, mobile guards are no more powerful than point guards.)

1.1.6 Polygon visibility

The concepts introduced in the previous section from the art gallery literature lead naturally to different notions of polygon visibility. Our work in sensor configuration planning involves visibility computations, in order that robust sensing strategies can be computed in complex geometric environments. The following definitions will be used throughout the thesis:

Definition 1.1 *A point p is* visible *from point q in an environment of obstacle polygons $\mathcal{P}$ if the segment $\overline{pq}$ does not cross the interior of any of the obstacles of $\mathcal{P}$.*

Definition 1.2 *A point p inside a polygon P is* visible *from point q if the line segment $\overline{pq}$ lies completely inside P.*

Note that $\overline{pq}$ may touch the boundary of P.

Definition 1.3 *A polygon P is said to be* completely visible *from a point $x \in P$ if every point on the boundary of P is visible from x.*

The *kernel* of a polygon P is the set of all points of P from which every point of P is visible. An equivalent formulation to Definition 1.3 is that a polygon P is completely visible from $x \in P$ if x lies in the kernel.

Extending Definition 1.3 to edges, we have the following:

Definition 1.4 *A polygon P is said to be* completely visible *from an edge e of P if each point of P is visible to every point along e.*

Equivalently, P is completely visible from an internal edge e if e is contained in the kernel of P. Figure 1.20 illustrates complete visibility from an edge.

Definition 1.5 *A polygon P is* weakly visible *from an internal edge e if each point of P is visible to at least one point along e.*

See Figure 1.21 for an illustration of weak visibility from an edge.

1.2 Dealing with uncertainty and complexity

In the previous section we reviewed some of the basic tools employed throughout the thesis. Here we introduce some of the more specialized techniques used to deal with the inherent *uncertainty* and *complexity* that are part of the problems we address in the thesis.

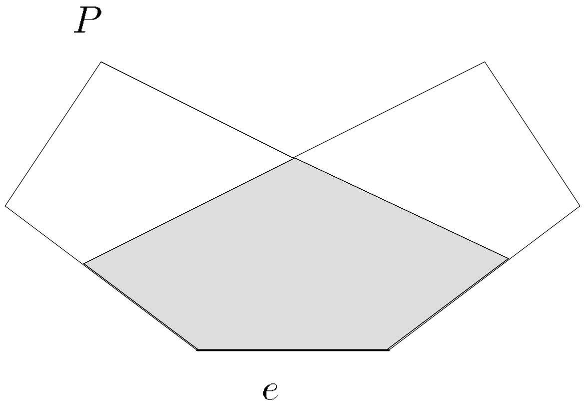

Figure 1.20: Polygon P is completely visible from edge e. The kernel of P is the shaded region.

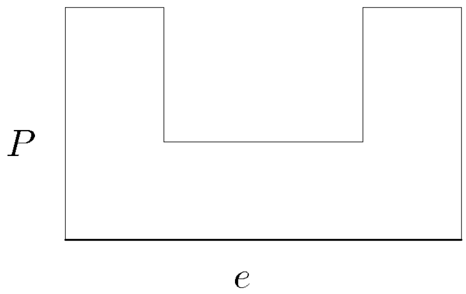

Figure 1.21: Polygon P is weakly visible from edge e.

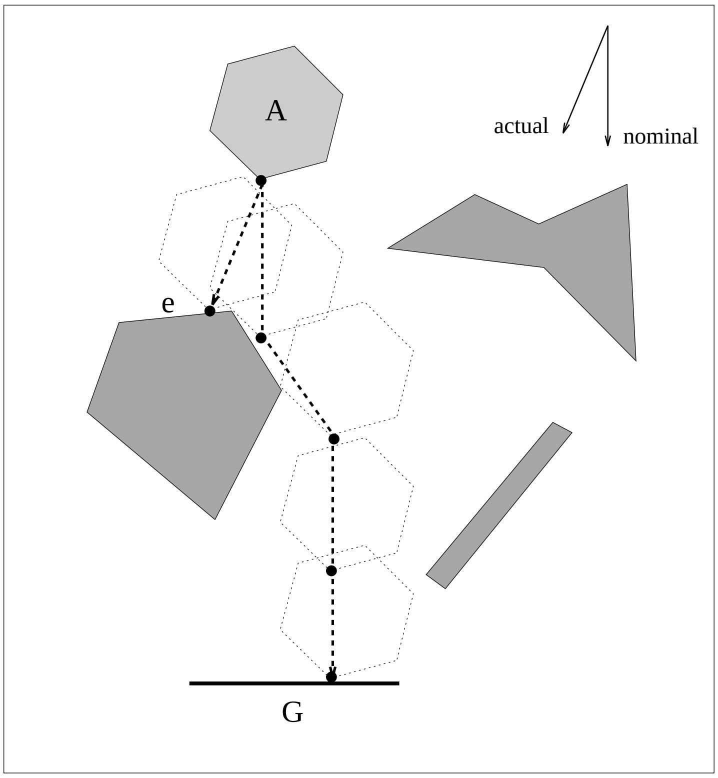

Figure 1.22: An illustration of a motion strategy which is subject to control uncertainty. A represents a robot that should maneuver through the obstacles moving in the direction labeled "nominal" and stop on the goal, labeled G. Clearance between the obstacles is tight, and may lead to A stopping prematurely on an obstacle edge such as e.

1.2.1 Uncertainty

Uncertainty in robot planning arises from errors in robot control and sensing, and in the geometric models of the task environment. Since real robots must successfully execute tasks in the presence of uncertainty, we must take uncertainty into account when planning robot strategies. See Figure 1.22 for an example of a motion strategy which is subject to uncertainty.

To deal with uncertainty in control, it is convenient to allow compliance on obstacle surfaces, where a *compliant motion* is one during which the robot may slide or stick on obstacle surfaces. For example, the successful execution path depicted in Figure 1.22 involves the robot sliding along the boundary of one of the obstacles. The robot control system employed must be capable of executing such motions. A hybrid force/position control system or one that implements a generalized damper could be used for executing compliant motions.

We model uncertainty in control geometrically with a cone centered at the

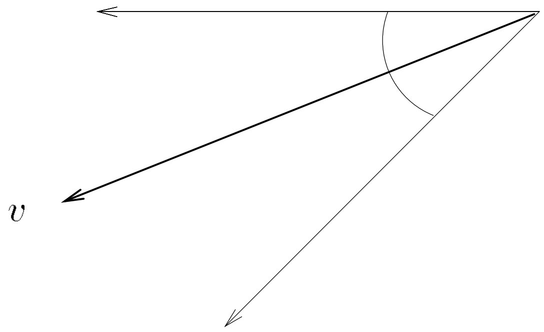

Figure 1.23: Velocity uncertainty cone about the commanded velocity v. The cone models the range of possible actual velocity directions at execution time, given that v is the commanded velocity.

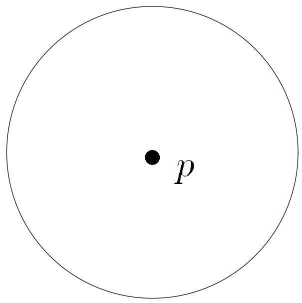

Figure 1.24: Positional uncertainty disc about the sensed position p. The disc models the set of possible actual positions, given that p is the sensed position.

commanded velocity, as in Figure 1.23. Positional uncertainty is modeled as a circle about the sensed position, as in Figure 1.24.

1.2.2 Backprojections

Given the above geometric representations for sensing and control uncertainty, we can reason geometrically about goal reachability. The *backprojection* of a goal is a region in configuration space from which particular motions are guaranteed to reach a goal [Erd84,Erd86]. Given a commanded velocity direction θ, the backprojection of the goal G with respect to θ is the set of all positions from which any trajectory consistent with the control uncertainty is guaranteed to reach G. See Figure 1.25 for an illustration of the backprojection for a particular commanded motion direction θ. In Chapter 2 we investigate how the backprojection changes with θ. We use this characterization to develop an efficient algorithm for motion planning with uncertainty.

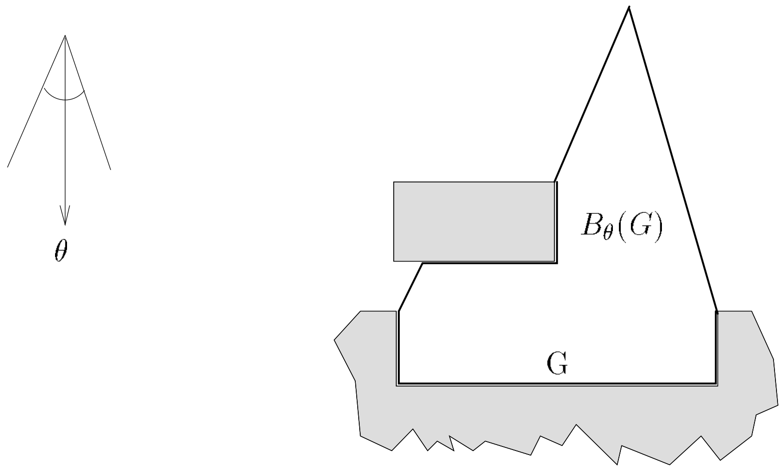

Figure 1.25: The backprojection of goal G for motion direction θ.

1.2.3 Recognizability regions

Given a complex geometric task environment, we must plan sensor strategies in a principled way. We introduce the notion of a *recognizability region* to characterize geometrically the set of sensor placements from which a robot can be monitored. For a region X of the robot configuration space, we define the *recognizability region* of the robot in X to be the set of all sensor configurations from which the sensor can detect the robot when its reference point lies in X. We compute recognizability regions using extensions of the visibility models introduced in Section 1.1.6. For example, Figure 1.26 shows the region from which a robot A is completely visible. Figure 1.27 shows the region from which A is partially visible. In Chapter 3 we present algorithms for computing these regions.

1.3 Review of related work

The contributions of this thesis forge connections between robotics, active sensing, and computational geometry. Related research in these areas is described below.

1.3.1 Classical motion planning

An algorithm for planning shortest collision-free paths using the visibility graph was first described by Lozano-Pérez and Wesley [LW79]. Extensions to this work, including the planning of shortest paths between a start and goal vertex, were described by Lee and Preparata [LP84] and Sharir and Schorr [SS86].

Schwartz and Sharir gave the first motion planning algorithms for the general algebraic case [SS83a,SS83b,SS83c,SAS84,SS84]. The technique used involved

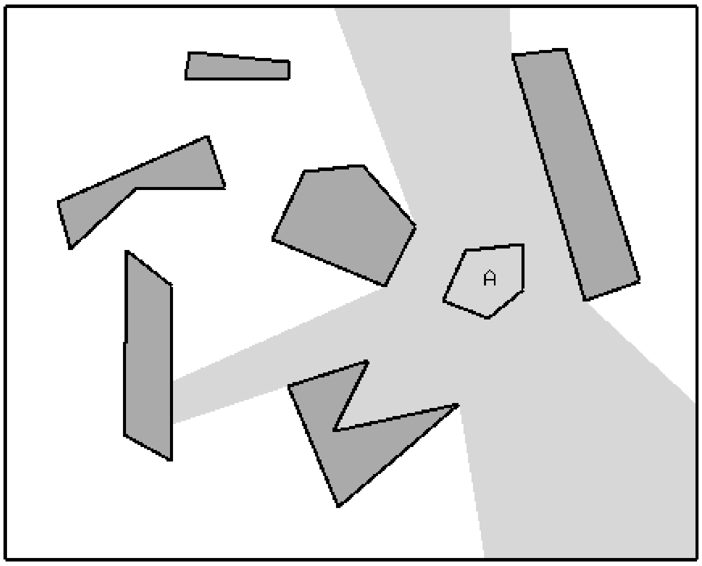

Figure 1.26: The recognizability region of robot A (under a complete visibility model) is shown lightly shaded.

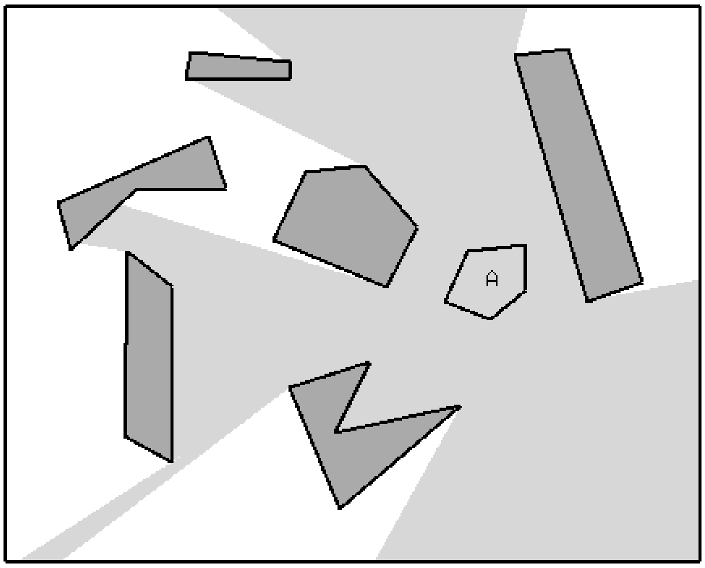

Figure 1.27: The recognizability region of robot A (under a partial visibility model) is shown lightly shaded.

decomposing the space of free placements into connected components and then searching a connectivity graph on these components. For an environment of geometric complexity n and a system with r degrees of freedom, the running time of their general solution was $O(n^{2^{r+6}})$. Reif gave the first lower bounds, demonstrating the problem to be PSPACE-hard when the number of degrees of freedom is specified as part of the input [Rei79].

Canny addressed the general motion planning problem and showed that a path can be found in time polynomial in the geometric and algebraic complexity of the obstacles, and singly-exponential in the degrees of freedom [Can87]. However, the *shortest*-path problem for a point among polyhedral objects in three dimensions has been shown to be NP-hard [CR87].

In addition to the decomposition-based approaches, other methods for solving the findpath problem are based on the idea of retraction and the use of the Voronoi diagram. The *Voronoi diagram* generated by a set of points in a Euclidean space partitions the space into convex regions, each having a single nearest source point under some metric. The generalized Voronoi diagram, introduced by Lee and Drysdale [LD81], is defined for points and line segments in the plane. Here, the regions generated are not necessarily convex. The advantages of the Voronoi diagram method for motion planning are that the diagram has lower complexity than the original space and can be efficiently searched for motion paths, and these paths will be maximally clear of obstacles. For an illustration, see Figure 1.28, which shows the Voronoi diagram for the configuration space of Figure 1.14. To find a path between two points in the diagram, it suffices to find paths from both points onto the diagram, and then search the diagram for a path to connect these two subpaths. Many motion planning algorithms have been based on this construction [OY85,OSY86,OSY87,LS87]. Canny and Donald [CD88] give an alternate formulation with lower algebraic complexity, allowing more efficient searches in high dimensional spaces.

The configuration space approach to path planning was first introduced by Lozano-Pérez [Loz83]. Brooks and Lozano-Pérez describe an implemented planner based on a subdivision of configuration space for a planar polygonal object with two translational and one rotational degrees of freedom [BL83]. Donald gives a method for constructing the configuration space for the full six degree-of-freedom motion planning problem, and describes an implemented path planner based on this algorithm [Don87].

See Latombe's book for a more detailed review of related research in robot motion planning [Lat91].

1.3.2 Motion planning with uncertainty

As mentioned above, much of the early work in motion planning made unrealistic assumptions about robot sensing and control. In more recent work in planning,

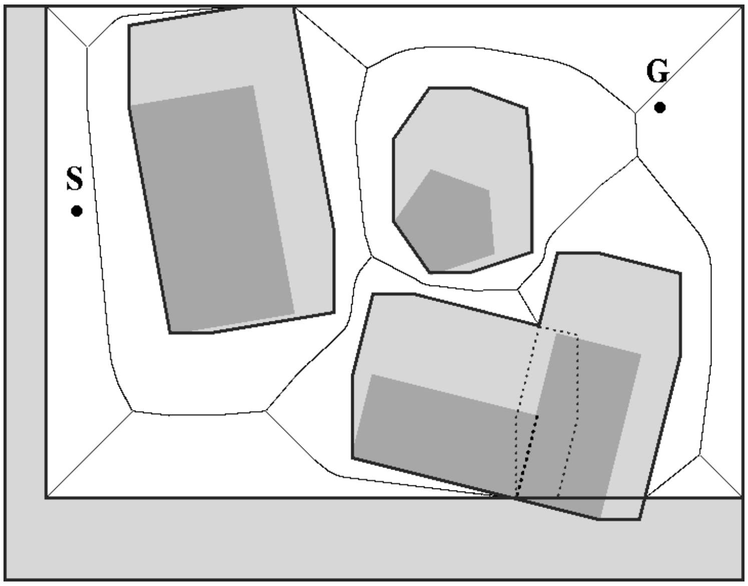

Figure 1.28: The Voronoi diagram for the configuration space of Figure 1.14. The thinly drawn curves are the edges of the Voronoi diagram. A path from S to G can be found by finding paths from both S and G onto the diagram, and then searching the diagram for a path to connect these two subpaths.

the assumptions of perfect robot control and sensing are relaxed, resulting in more robust solutions.

Lozano-Pérez, Mason and Taylor have presented a formal system for incorporating sensing and control uncertainty into the planning process [LMT84]. We refer to their system as the *LMT framework for fine motion planning.* They develop the notion of a *preimage* to characterize the set of locations from which goal attainment is recognizably reachable. See Figure 1.29 for an illustration of a simple preimage. Preimages are not computable in general [LMT84,Erd86].

See Erdmann's master's thesis [Erd84] for an excellent review of the underlying models and tools employed in the LMT framework, including a thorough treatment of configuration space and generalized damper dynamics.[1]

Erdmann has developed the notion of a *backprojection* as a computable approximation to preimages [Erd84,Erd86]. The backprojection of a goal G with respect to a commanded velocity direction θ is the set of all positions from which G is guaranteed to be reached, *i.e.*, the set from which all possible trajectories consistent with the control uncertainty are guaranteed to reach G. This work has separated

[1]The *generalized damper* is a dynamical model specified by the relationship $F = B(v - v_0)$ between forces and velocities, where F is the vector of forces and torques acting on a moving object, v_0 is the commanded velocity, v is the actual velocity, and B is a damping matrix (generally taken to be the identity matrix). For more details, see Chapter 2 of this thesis, as well as [Whi77, Mas81,Erd84,Don89].

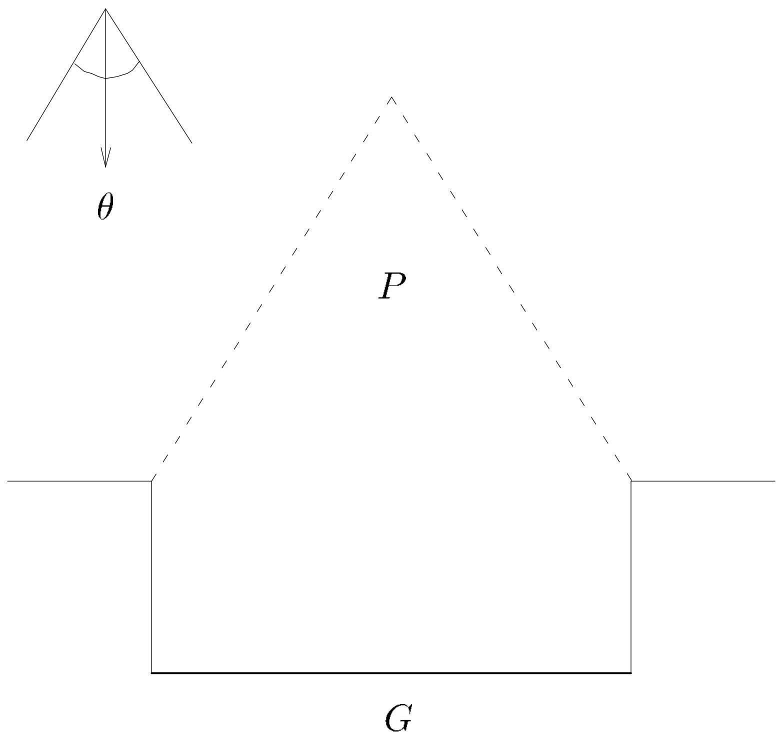

Figure 1.29: P is the preimage of the goal G with respect to commanded velocity direction θ and the control uncertainty cone shown.

the motion planning problem into questions of *reachability* and *recognizability*.

Shekhar and Latombe analyze the interaction between goal reachability and goal recognizability in motion planning with uncertainty [SL91]. They formally characterize the recognition power of different motion termination conditions, and show that the recognition power of a termination condition can be augmented by giving the predicate knowledge of itself.

Friedman, Hershberger, and Snoeyink have considered the problem of planning compliant motion within a simple polygon [FHS89]. In a later paper they present an input-sensitive algorithm for finding a one-step motion strategy in a polygon with k polygonal holes [FHS90].

We mention here a few of the theoretical results characterizing the complexity of robot motion planning with sensing and control uncertainty. Canny and Reif [CR87] have shown that, in three dimensions, the problem of one-step motion planning is NP-hard and the problem of multi-step motion planning is NEXPTIME-hard. Previously, Natarajan had shown the multi-step problem to be PSPACE-hard [Nat86]. Canny has also given doubly-exponential upper bounds for the $3D$ translational multi-step problem [Can89].

In addition to sensing and control error, a third type of uncertainty is due to *model error*. Donald extends the LMT framework to incorporate model error. His work provides a framework for computing motion strategies that are guaranteed to succeed in the presence of sensing, control, and geometric model uncertainty [Don89]. In earlier work, Brooks presented a symbolic error analysis system for use in a plan checker; model uncertainty was one of the sources of error considered.

The *Error Detection and Recovery (EDR)* system of Donald [Don89] provides a framework for constructing manipulation strategies when guaranteed plans cannot be found or do not exist. An EDR strategy attains the goal when the goal is recognizably reachable, and signals failure otherwise. We examine an application of our work in sensor configuration planning to the EDR framework in Chapter 4.

Another extension of the basic motion planning definition is the simultaneous consideration of dynamic and kinematic constraints on motion trajectories. The idea of *kinodynamic planning* was introduced by Canny, Donald, Reif and Xavier [CDRX88,DXCR93]. The idea is that planned motions should avoid all obstacles by a speed-dependent safety margin, giving rise to the idea of a time-optimal motion plan, or an optimal kinodynamic trajectory. Xavier and Donald have given provably good approximation algorithms for solving a class of optimal kinodynamic planning problems [Xav92,DX94a,DX94b].

For other related research on planning in the presence of uncertainty, see the grasp planning work of Brost [Bro85,Bro86] and the preimage backchaining work of Latombe [Lat88].

1.3.3 Information requirements for robot tasks

Much attention has been given recently to the question of how much *information* is required to accomplish a task. Donald has developed a theory of *information invariants* to characterize the information complexity of robot tasks [Don93a, Don93b]. This work investigates the trade-offs between sensing, action, communication, and computation. Donald, Jennings, and Rus build on the information invariants framework to explore the information requirements of several manipulation tasks. They describe implemented manipulation protocols for pushing, reorientation of large objects, and cooperative pushing [DJR93a,DJR93b,DJR94]. In earlier work, Donald and Jennings propose a framework of *constructive recognizability* to simplify robot programming in uncertain environments [DJ92a,DJ92b]. In related work, Erdmann [Erd93] proposes a method for automatically designing sensors based on the information requirements of a robot task. Jia and Erdmann [JE94] present algorithms and lower bounds for the problem of locating the minimum number of sensing points required to distinguish between a finite set of polygonal shapes. Canny and Goldberg [CG93] consider robot systems with simple sensors such as light beams, and simple actuators such as parallel-jaw grippers. They call their paradigm RISC, for *reduced intricacy in sensing and control*. Their framework is primarily intended for repetitive operations in known, structured environments.

We consider our work in sensor planning to be similarly motivated to the research agendas described above. In particular, we are interested in configuring sensors based on task requirements. Our contribution here is the method we develop for characterizing the information returned by sensors.

1.3.4 Active sensing

Researchers in robotics and machine vision have explored many aspects of the sensor configuration problem. Hager and Mintz [HM91] define the sensing problem to be that of constructing a consistent geometric description (shape, size, and position) of objects. They present a mathematical framework for describing geometric sensing problems and investigate the cost of acquiring information. To compute optimal sensing strategies, they employ the Bayesian decision-making theory using a grid-based representation to approximate probability distributions.

Cameron and Durrant-Whyte present a Bayesian approach to optimal sensor placement [CD90]. Rimey and Brown describe a task-oriented vision system for information gathering that uses Bayes nets [RB92,BY92].

Krotkov and Bajcsy have addressed the problem of reliably measuring the three-dimensional position of objects in an unknown and cluttered scene. They use cooperative range recovery techniques to support mapping [KB93]. Other researchers have described methods for acquiring $3D$ information given range images obtained from a light stripe range finder [MB92,KK94]. The sensor placement problem has also been addressed for visual tracking and vision-guided exploration [NK94, KDL94].

Hutchinson [Hut91] introduces the concept of a *visual constraint surface* to control motion. The idea is to combine position, force, and visual sensing in order to produce error-tolerant motion strategies. His work builds on that of preimage planners by adding visual feedback to compensate for uncertainty. Details on the implementation of vision-based control are described in a subsequent paper with Castaño [CH92].

Sharma and Hutchinson [SH94] define a measure of *robot motion observability* based on the relationship between differential changes in the position of the robot to the corresponding differential changes in the observed visual features. Lacroix, Grandjean, and Ghallab [LGG92] describe a method for selecting view points and sensing tasks to confirm an identification hypothesis.

Cowan and Kovesi [CK88] study the problem of automatic camera placement for vision tasks. They consider the constraints on camera location imposed by resolution and focus requirements, visibility and view angle, and forbidden regions depending on the task. Given values bounding these constraints, they compute the set of camera locations affording complete visibility of a surface in $3D$. Zhang [Zha92] considers the problem of optimally placing multiple sensors.

Ellis [Ell92] presents an approach for acquiring tactile data to accomplish model-based object recognition. The problem he addresses is the tactile equivalent of choosing a position and orientation for a camera or ranging device.

The paradigm of *active vision* encompasses the problems of attention focusing, gaze control, eye-hand coordination, and combining vision with behavior. For information on research in active vision, see the survey article edited by Swain and Stricker [SS93], and the books by Blake and Yuille [BY92], Ballard and Brown [BB82], and Marr [Mar82]. Ballard [Bal91] has also introduced the notion of *animate vision* to tie visual behavior into the larger context of behavior.

1.3.5 Computational Geometry

A number of researchers have given worst case optimal algorithms for computing the visibility graph of a set of polygons in the plane [AAG+86,Wel85]. Ghosh and Mount [GM87] give an output sensitive $O(E + n \log n)$ algorithm, where E is the number of edges in the visibility graph and n is the total number of vertices in all the input polygons. In the worst case, $E = O(n^2)$.

The questions of detecting polygon visibility and constructing visibility regions under a variety of assumptions is a rich area of past and ongoing research in computational geometry. We mention here a few of the papers most closely related to our problem. Suri and O'Rourke [SO86,O'R87] give an $\Theta(n^4)$ algorithm for the problem of computing the locus of points weakly visible from a distinguished edge in an environment of line segments. Their lower bound of $\Omega(n^4)$ for explicitly constructing the boundary of the weak visibility region holds as well for our computation of recognizability regions under a weak visibility assumption. Bhattacharya, Kirkpatrick and Toussaint [BKT89] introduce the concept of *sector visibility* of a polygon, and give $\Theta(n)$ and $\Omega(n \log n)$ bounds, depending on the size of the visibility wedge, for determining if a polygon is weakly externally visible. The problem of planar motion planning for a robot with bounded directional uncertainty is considered by de Berg *et al.* [dBGH+93]. They give algorithms for constructing the regions from which goals may be reached, and show that the complexity of the regions depends on the magnitude of the uncertainty angle.

Teller [Tel92] solves the weak polygon visibility problem for a special case in $3D$. Namely, he computes the antipenumbra[2] of a convex area light source shining through a sequence of convex areal holes in three dimensions. See Figure 1.30 for illustrations. For an environment of total edge complexity n, he gives an $O(n^2)$ time algorithm for computing the piecewise-quadratic boundary of the antipenumbra, which will be non-convex and disconnected in general. Computation of the antiumbra, the volume from which all points on the light source can be seen, is analogous to the concept of complete visibility in computational geometry. Teller notes that for an environment consisting of a sequence of transparent convex holes, the antiumbra is convex, polyhedral, has complexity $O(n)$, and can be computed in time $O(n \log n)$.

Tarabanis and Tsai [TT92] examine the question of complete visibility for general polyhedral environments in $3D$. They present two methods for computing the locus of all viewpoints from which a fixed feature polygon can be entirely seen in an environment of polyhedra. Their "boundary-based" approach computes an arrangement of size $O(m^3 n^3)$ for a feature polygon of size m and a polyhedral environment of size n, and then classifies the cells as to whether they are part of the visibility region, giving an overall worst case complexity of $O(m^4 n^4)$. Their "decomposition-based" algorithm computes an arrangement of triangles and runs in overall time $O(m^3 n^3)$.

Guibas, Motwani and Raghavan consider an abstraction of the robot localization problem [GMR92]. Given a simple polygon P (representing the map of a known environment) and a star-shaped polygon V (representing the portion of the map visible from the robot's position), the problem is to find a point or set of points in P from which the portion of P that is visible is congruent to V (*i.e.*, given V, the robot must determine its position in the map). They give a method of preprocessing P so that subsequent queries V can be answered in optimal time in the size of the output.

[2]The *antipenumbra* is the volume from which some, but not all, of a light source can be seen.

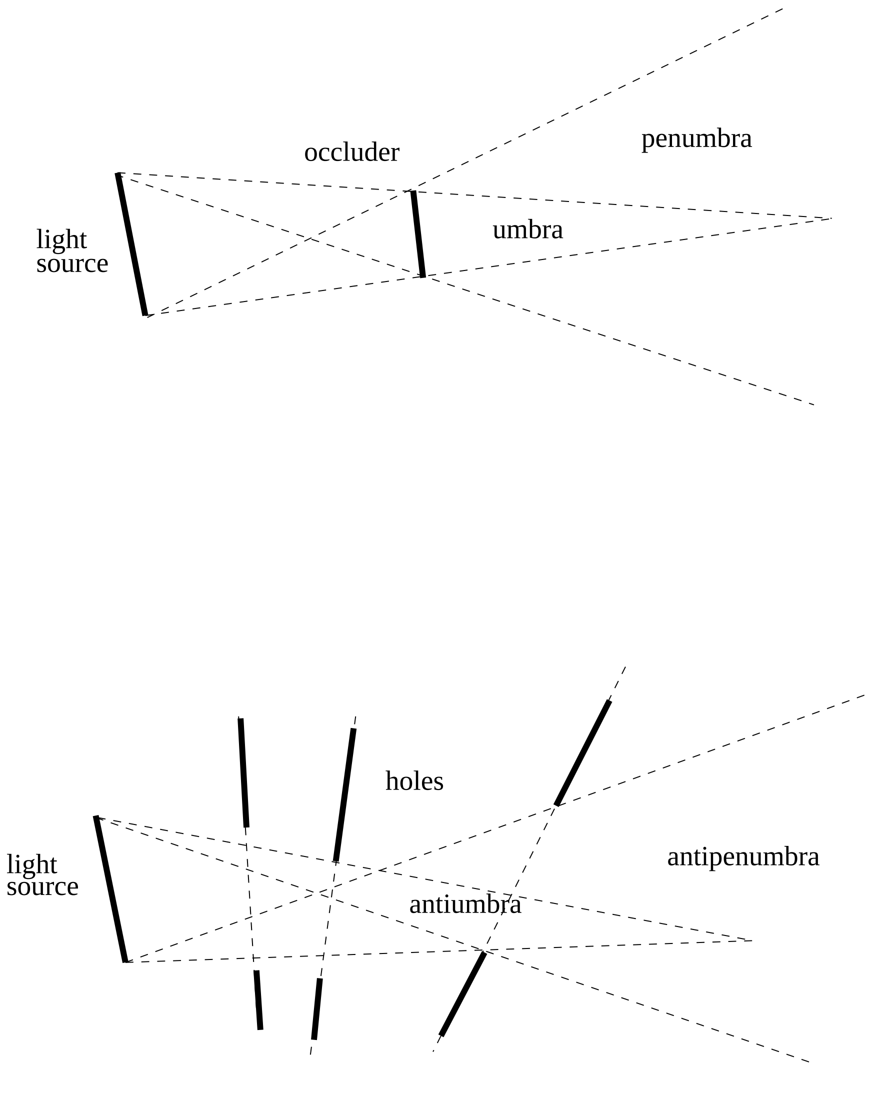

Figure 1.30: Illustrations of the umbra, penumbra, antiumbra, and antipenumbra.

1.4 Thesis contributions

This thesis addresses foundational problems in robot motion planning and sensor configuration planning. Its contributions are algorithmic, theoretical, and in systems development. In this section we give a brief overview of our results. A table summarizing our algorithmic contributions is given at the end of the section.

1.4.1 Motion planning

The *one-step compliant motion planning problem* is to find a single translational motion direction θ that guarantees reaching a goal from a specified start region. Robot control uncertainty must be explicitly taken into account. Our algorithm for this problem extends and improves the $O(n^4 \log n)$ algorithm by Donald [Don88, Don90] in a planar environment with n vertices. Recall that the backprojection $B_\theta(G)$ is defined for goal G and commanded motion direction θ to be the set of all points from which the goal is guaranteed to be reached (see Figures 1.25 and 1.31). The *non-directional backprojection* is defined over all θ to be the region swept out by the backprojection as θ changes. A guaranteed motion plan is one for which the goal is guaranteed to be reached from any position in the start region. A *one-step* motion plan is a single commanded motion direction that guarantees reaching the goal. See Figure 1.31 for an illustration. Our motion planning algorithm computes the non-directional backprojection in order to find a motion direction θ for which the start region is contained in the backprojection $B_\theta(G)$. Details and more precise definitions will be given in Chapter 2. We give a precise characterization of the non-directional backprojection that leads to an efficient algorithm for one-step motion planning.

We show the following results:

Theorem 2.3 *There are $O(n^2)$ changes to the topology of the backprojection over all values of θ.*

Theorem 2.4 *Given a goal G of constant size and an arrangement of input polygons $\mathcal{P}$ of size $O(n)$, a representation of size $O(n^2)$ for the non-directional backprojection $B(G)$ can be computed in time $O(n^2 \log n)$.*

Theorem 2.6 *The one-step planar compliant motion planning problem with uncertainty can be solved in time $O(n^2 \log n)$.*

Our contributions are both in characterizing the complexity of planning with uncertainty and in providing efficient algorithms. The results presented here have appeared in [Bri92] and an earlier version of this work appeared in [Bri89].

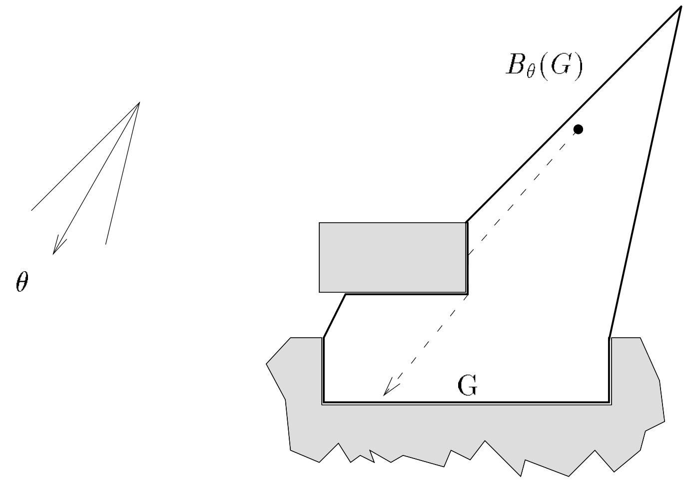

Figure 1.31: The dashed line represents a one-step motion plan consistent with motion direction θ. The backprojection of the goal is the region $B_\theta(G)$.

1.4.2 Sensor configuration planning

In the area of sensor configuration planning, our contribution is to bridge the gap between research on the theoretical problems of polygon visibility and research on sensor configuration which assumes simple geometric environments. In particular, our work is the first to consider weak visibility as a model for detectability. In cluttered environments, requiring complete visibility may restrict sensor placement too severely, while allowing weak visibility can greatly increase the set of sensor positions from which the robot is detectable. Under both models of detectability, our work is the first to precisely characterize the complexity of computing goal recognizability, and to compute visibility maps for a robot at any position within a goal polygon.

Figure 1.32 shows an example environment, in which the reference point of a robot A lies inside a goal region G. Point p is an example sensor placement from which A is visible when in G. Figure 1.33 shows the partial visibility region for robot A at a fixed position, and Figure 1.34 shows the complete visibility region for A at the same position.

Our results in this area are the following:

Theorem 3.1 *The recognizability region of a robot translating and rotating through a goal with k vertices can be computed in time $O(n\alpha(n) + nk)$ in an environment with n vertices in the complete visibility model.*

Theorem 3.2 *In the weak visibility model, the recognizability region of a robot translating through a goal of size k can be computed in time $O(kmn^3(n+m))$ for an environment of complexity n and a robot of complexity m.*

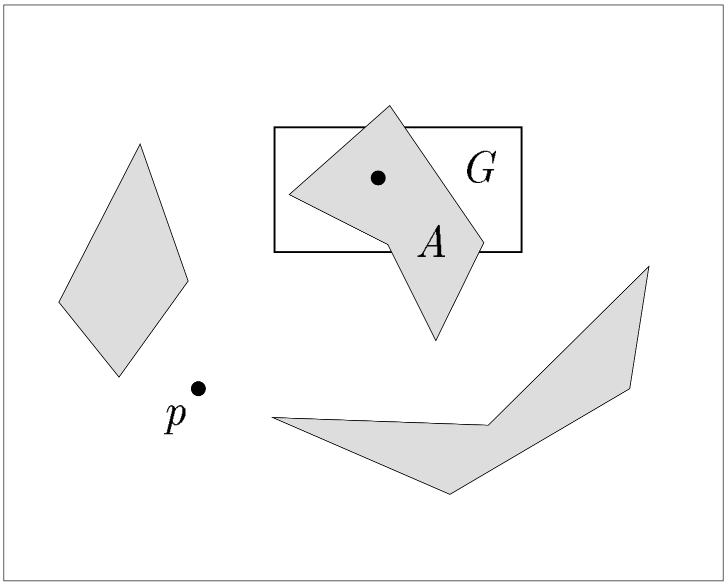

Figure 1.32: An example environment for the sensor configuration problem. A is the robot, with its reference point at a position in the goal region G. The other polygons are obstacles. Point p is an example sensor placement from which A is always visible when in G.

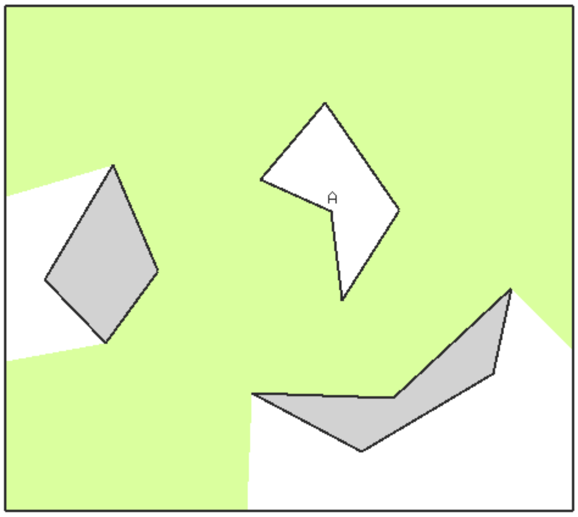

Figure 1.33: Illustration of the partial visibility region of the robot A.

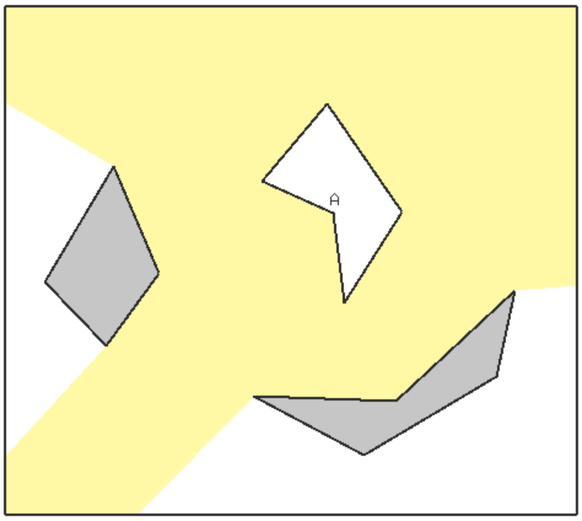

Figure 1.34: Illustration of the complete visibility region of the robot A.

Some of the work presented here in the area of sensor configuration planning appeared in [BD94].

We have applied our techniques for computing recognizability regions to the *Error Detection and Recovery Framework* [Don89]. We sketch how our algorithms for computing visibility regions can be applied to the problem of distinguishing between success and failure in EDR planning.

1.4.3 Systems, implementations, and experiments

A further contribution is the systems we have developed for sensor configuration planning. We have implemented our algorithms described in Chapter 3 for computing polygon visibility regions. We used our implementation of the complete visibility algorithm for a stationary robot (described in Section 3.3.1) to conduct experiments with two of the mobile robots in our robotics and vision laboratory. This led to the development of a robot surveillance demonstration system, which we describe in Chapter 5. We use a mobile robot called TOMMY, equipped with a CCD camera, as the sensor platform. TOMMY's task is to monitor a particular region of the lab and announce when another mobile robot, LILY, enters the region. Given a map of the robotics lab, our system computes a new sensor configuration that affords TOMMY complete visibility of LILY within the region of interest. A motion plan is then computed and executed so that TOMMY can monitor the region.

Table 1.1: Motion planning and sensor configuration planning algorithms. In the complexity bounds given, n is the number of vertices in the environment, m is the number of vertices of the robot, and k is the number of vertices of the goal.

Chapter	Algorithm	Algorithm Complexity
2	One-Step compliant motion planning with uncertainty	$O(n^2 \log n)$
3	Complete visibility of a stationary robot	$O(n\alpha(n) + nm)$
3	Complete visibility of a translating robot in a goal region	$O(n\alpha(n) + n(m + k))$
3	Complete visibility of a translating and rotating robot in a goal region	$O(n\alpha(n) + nk)$
3	Weak visibility of a stationary robot	$O(n^2(n + m)^2)$
3	Weak visibility of a translating robot in a goal region	$O(kmn^3(n + m))$

1.4.4 Summary of algorithmic contributions

Table 1.1 summarizes the algorithmic contributions of the thesis.

The complexity bound given in the table for computing the weak visibility region of a stationary robot is for explicitly computing the boundary of the visibility region. The region can be given as a union of visibility triangles in time $O(n(n+m))$. The incorporation of sensor placement and aim uncertainty into this computation is discussed in Section 3.5. We show that the consideration of sensor aim uncertainty in the weak visibility model lowers the complexity of explicitly computing the boundary of the visibility region to $O(n(n + m))$.

Our motion planning algorithm gives a quadratic-time improvement over the best previous result for this problem. Our algorithms for computing goal recognizability are the first such results, and do not represent improvements over previous work. This work is the first to precisely characterize the complexity of computing goal recognizability. Our algorithms for computing visibility regions for a stationary robot are also the first such algorithms to be implemented and used with mobile robots for sensor configuration planning.

1.5 Outline of thesis

In Chapter 2 we give a precise characterization of the non-directional backprojection and present our algorithm for one-step motion planning in the plane.

In Chapter 3 we begin our investigation of sensor configuration planning. Algorithms are presented here for monitoring a moving polygon in both a weak and complete model of visibility.

We sketch an application of our work in sensor configuration planning to the EDR framework in Chapter 4.

In Chapter 5 we discuss the implementations of our algorithms for computing polygon visibility. A detailed description of the robot surveillance project is also provided.

We conclude in Chapter 6 with a summary and a discussion of some directions for future work.

Chapter 2

Motion Planning with Uncertainty

Motion plans for real robots must account for errors during execution. Consequently, given bounds on errors in sensing and control, we would like to plan motions that are guaranteed to succeed even in the worst case. *Uncertainty* as to control fundamentally changes the complexity of motion planning and the techniques employed. The introduction of uncertainty leads naturally and necessarily to allowing the robot to contact and comply with obstacle surfaces, because doing so greatly enriches the set of problems that can be solved.

We address the concrete and basic problem of finding a single commanded motion direction to maneuver a point robot from an uncertain start position in the plane to a specified goal where the robot is guaranteed to stop. Motions are strictly translational; thus the problem, for example, of modeling the geometric interactions of a peg and a hole can be reduced to navigating a point in configuration space [Loz83]. As in the classical motion planning problem discussed in Chapter 1, we assume the problem is posed in an environment of planar polygonal obstacles that is known and can be modeled exactly. The realization of a command, however, is subject to uncertainty since robots have imprecise sensing and imperfect control and therefore can only execute commands to within a given accuracy. Given a bound ϵ_c on the control error, we model this error as a cone in configuration space about a vector in the direction of the commanded motion θ and call it the *uncertainty cone* $U_{\epsilon_c}(\theta)$. See Figure 2.1 for an illustration. While executing a motion plan, the robot complies with the environment and may choose any direction consistent with the commanded direction and the control uncertainty. The direction chosen may vary over the execution of the plan. In what follows, n denotes the number of vertices in the environment and E denotes the number of edges in the visibility graph, where we assume $E = \Omega(n)$.

To deal with uncertainty in control, we allow *compliance* on obstacle surfaces, where a *compliant* motion is one during which the robot may slide or stick on

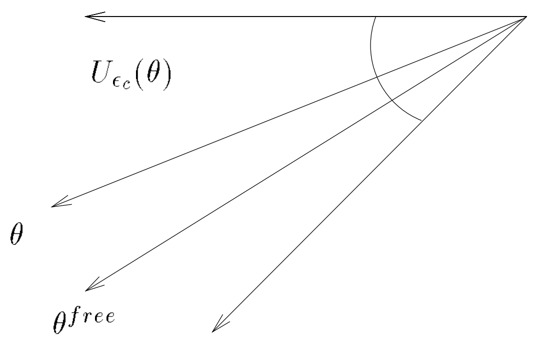

Figure 2.1: Uncertainty cone $U_{\epsilon_c}(\theta)$. The actual direction θ^{free}, chosen by the control system, may be any direction in the cone.

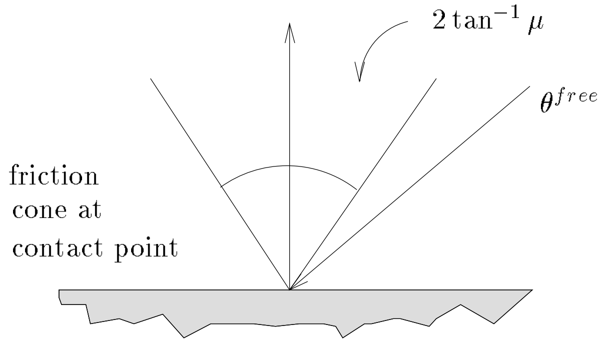

Figure 2.2: Sliding occurs at motion direction θ^{free}.

obstacle surfaces. We model this effect by assuming generalized damper dynamics [Whi77,Mas81,Erd84,Don89] and Coulomb friction at point contacts, where the coefficient of friction μ is known and remains fixed for the environment. To determine if an obstacle surface is a sticking surface at motion direction θ, we check whether a vector at direction θ lies inside the negative friction cone at the point of contact. If the vector lies outside the negative friction cone, then sliding will occur, otherwise sticking may occur (see Figure 2.2). To determine if sticking can occur on a vertex, we assume that the vertex can produce reaction forces that are linear combinations of the reaction forces that the adjacent edges can produce [Erd84, Erd86]. In this model, we take a worst case approach; that is, we assume that if sticking is possible at a point, then the motion plan must prevent the point from being reached unless the point is in the goal.

Our method involves the construction of a concise representation for a structure called the *non-directional backprojection* of the goal [Erd84,Erd86,Don88,Don90]. By analyzing the changes to the backprojection as the motion direction varies, we give a complete characterization of the non-directional backprojection and its complexity, and an efficient representation. To achieve this result, we develop an amortization strategy that is powerful enough to bound the number of changes to the boundary of the backprojection by $O(n^2)$. Hence, we can tighten the previous bound [Don88,Don90] on its combinatorial complexity from $O(n^3)$ to $O(n^2)$, and improve the algorithm for computing it from $O(n^4 \log n)$ to $O(n^2 \log n)$. As our examples show, the non-directional backprojection is a complex object that can undergo very large global changes at a single event while remaining locally monotonic. We obtain our result by characterizing this local monotonicity.

We state the problem as follows:

Definition 2.1 *Given a planar polygonal environment $\mathcal{P}$ with start region R and goal G, both having a constant number of vertices, the* one-step planar compliant motion planning problem *is to find a commanded motion direction θ such that any trajectory from R consistent with the control uncertainty ϵ_c is guaranteed to reach G. The path should avoid obstacles or comply with the environment.*

2.1 The directional backprojection

Given goal G, and commanded direction θ, the *backprojection* $B_\theta(G)$ is the set of all initial positions such that any trajectory consistent with the control uncertainty is guaranteed to reach the goal. Donald and Erdmann show that for constant size G, $B_\theta(G)$ can be computed in $O(n \log n)$ time using plane sweep techniques. Erdmann's algorithm is as follows [Erd86,Don89]:

1. For each non-goal vertex, determine whether the inverted control uncertainty cone $U_{\epsilon_c}(\theta)$ intersects the friction cone at that vertex. If so, call this vertex a *sticking* vertex under commanded motion θ.

2. On each sticking vertex, erect two constraint rays parallel to the edges of the inverted control uncertainty cone.

3. Compute the arrangement of the environment with these $O(n)$ additional constraint rays.

4. Starting at a point in the goal, trace out the backprojection region.

Figure 2.3 illustrates the construction.

We call the maximal initial segment in free space of a constraint ray erected on a sticking vertex a *free space edge*. The sticking vertex (called the anchor vertex) on which a free space edge e is erected remains fixed while the angle of e changes with θ. Recall that constraint rays lie parallel to one of the edges of the control uncertainty cone, which lie at angle $\theta \pm \epsilon_c$. A *left edge* anchored at p, denoted $\ell(p, \theta)$, is a free space edge lying at angle $\theta + \pi - \epsilon_c$ for motion direction θ and control uncertainty ϵ_c. A *right edge* anchored at q, denoted $r(q, \theta)$, is a free space

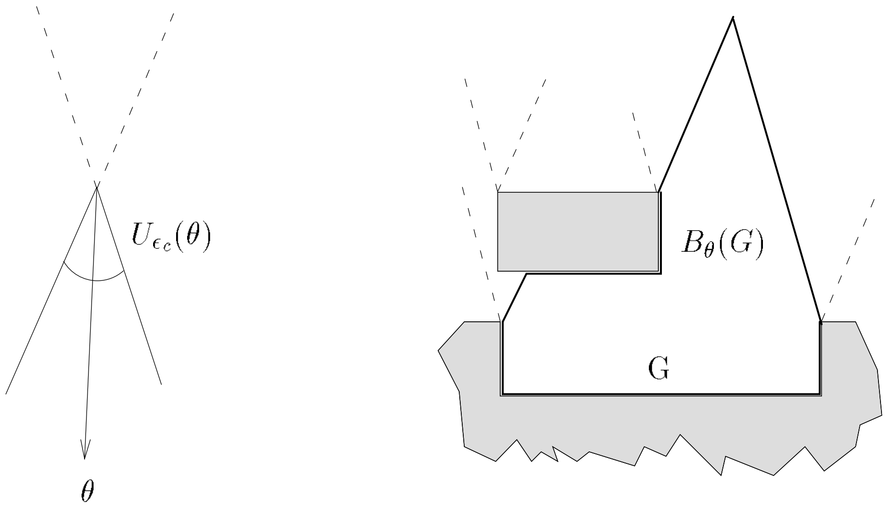

Figure 2.3: Backprojection construction.

edge lying at angle $\theta+\pi+\epsilon_c$. See Figure 2.4. If free space edge ℓ anchored at p and free space edge r anchored at $q \neq p$ intersect, we call the point of their intersection a *free space vertex*. Free space vertices are uniquely determined by their generating edges. A free space edge is *vgraph critical* when it lies coincident with an edge of the visibility graph and therefore joins two obstacle vertices in free space. We employ the convention that as θ increases monotonically over the range $[0, 2\pi)$, the corresponding control uncertainty cone $U_{\epsilon_c}(\theta)$ rotates in an anticlockwise direction, which has the effect of locally "rotating" the backprojection in an anticlockwise direction.

Lemma 2.1 *For any direction θ and goal G of constant size, the backprojection $B_\theta(G)$ has size $O(n)$.*

Proof: The environment $\mathcal{P}$ has n obstacle edges and n vertices, that in turn contribute $O(n)$ constraint rays when cones are erected on all the sticking vertices. The backprojection is built from obstacle edges and free space edges, where a free space edge is a constraint ray anchored at an obstacle vertex and intersecting an obstacle edge or another constraint ray. Since at most two free space edges are anchored at each obstacle vertex, the backprojection has size at most $O(n)$. $\square$

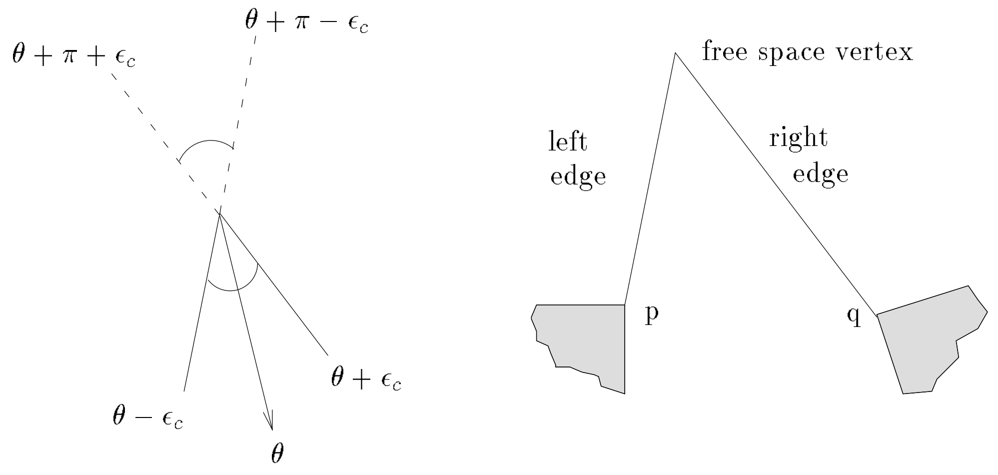

Figure 2.4: Free space vertex generated by the intersection of two free space edges. The left edge lies at angle $\theta + \pi - \epsilon_c$ and the right edge lies at angle $\theta + \pi + \epsilon_c$.

2.2 The non-directional backprojection

In order to find a commanded motion direction θ for which all trajectories from R are guaranteed to reach G, we evaluate the predicate $R \subset B_\theta(G)$ for selected values of θ. As we shall see, it suffices to consider those values of θ, called *critical values*, at which the topology of the backprojection changes. To this end, it is convenient to index the backprojection $B_\theta(G)$ with the critical motion direction θ at which it arises. This leads to the definition of the *non-directional backprojection* $B(G)$ as the set in $\mathbb{R}^2 \times S^1$

$$B(G) = \bigcup_\theta (B_\theta(G) \times \{\theta\}) .$$

Given that the topology of the backprojection does not change between critical events, Donald's approach [Don88,Don90] is to build $B(G)$ by computing a backprojection slice at each critical θ, and then test $B(G)$ for intersection with the cylinder $R \times S^1$ over the start region R. His algorithm builds a representation of size $O(n^3)$ for the non-directional backprojection in time $O(n^4 \log n)$. To obtain his result, Donald shows that we can bound the number of motion directions at which the topology of the backprojection changes, and thereby obtain a polynomial-sized representation for the non-directional backprojection. Since our proof allows this bound to be improved from $O(n^2)$ to $O(E)$, we give our own rendering.

Lemma 2.2 *There are $O(E)$ motion directions at which the topology of the backprojection changes.*

Proof: The topology of the backprojection may change when any of the following events occurs:

Sliding critical event The determination of sliding vs. sticking on an obstacle edge changes due to a change in the motion direction θ.

Vgraph critical event A constraint ray becomes coincident with a visibility edge.

Vertex critical event A free space vertex of the backprojection coincides with an obstacle edge.

Since the topology of the backprojection can change only when an edge is inserted or deleted, these are the only events that can cause a topological change.

Sliding critical events contribute $O(n)$ critical values of θ since the determination of sliding vs. sticking on an edge can change at most 4 times. The visibility graph has $O(E)$ edges, and since all constraint rays are parallel to one or the other of the two edges of the control uncertainty cone, there are $O(E)$ vgraph critical values of θ.

Vertex critical events contribute $O(E)$ critical values since we can charge each vertex critical event to a unique visibility edge or sliding event as follows. Suppose the free space vertex determined by left edge $\ell(p, \theta_0)$ and right edge $r(q, \theta_0)$ lies on obstacle edge e. Let x be the endpoint of e such that $\ell(p, \theta_0)$ has passed over x. Then one of the following must have occurred (see Figure 2.5):

1. Edge $\ell(p, \theta_1)$, $\theta_1 < \theta_0$, was coincident with the visibility edge between p and x.

2. Point x is not visible from p, and edge $r(q, \theta_2)$, $\theta_1 < \theta_2 < \theta_0$, was coincident with a visibility edge between q and some vertex on an obstacle between p and x.

3. No such vgraph event as in 1. or 2. occurred, in which case at least one of p and q has become a sticking vertex at θ_3, $\theta_1 < \theta_3 < \theta_0$.

We assume that the environment is in general position, that is, at most one critical event will occur at any θ. Under this assumption, one of these events will occur first. In other words, we can choose θ_i, $1 \leq i \leq 3$, for which $\theta_0 - \theta_i$ is minimized. Thus we can charge the vertex event to a vgraph or sliding event at θ_i. At most one other vertex event will be charged to the critical event at θ_i since, without any intervening sliding or vgraph events, a free space vertex travels in a piecewise circular arc and can coincide with an obstacle edge at most twice. We have shown above that the number of sliding and vgraph events is $O(E)$, so it follows that the number of vertex critical events is $O(E)$. □

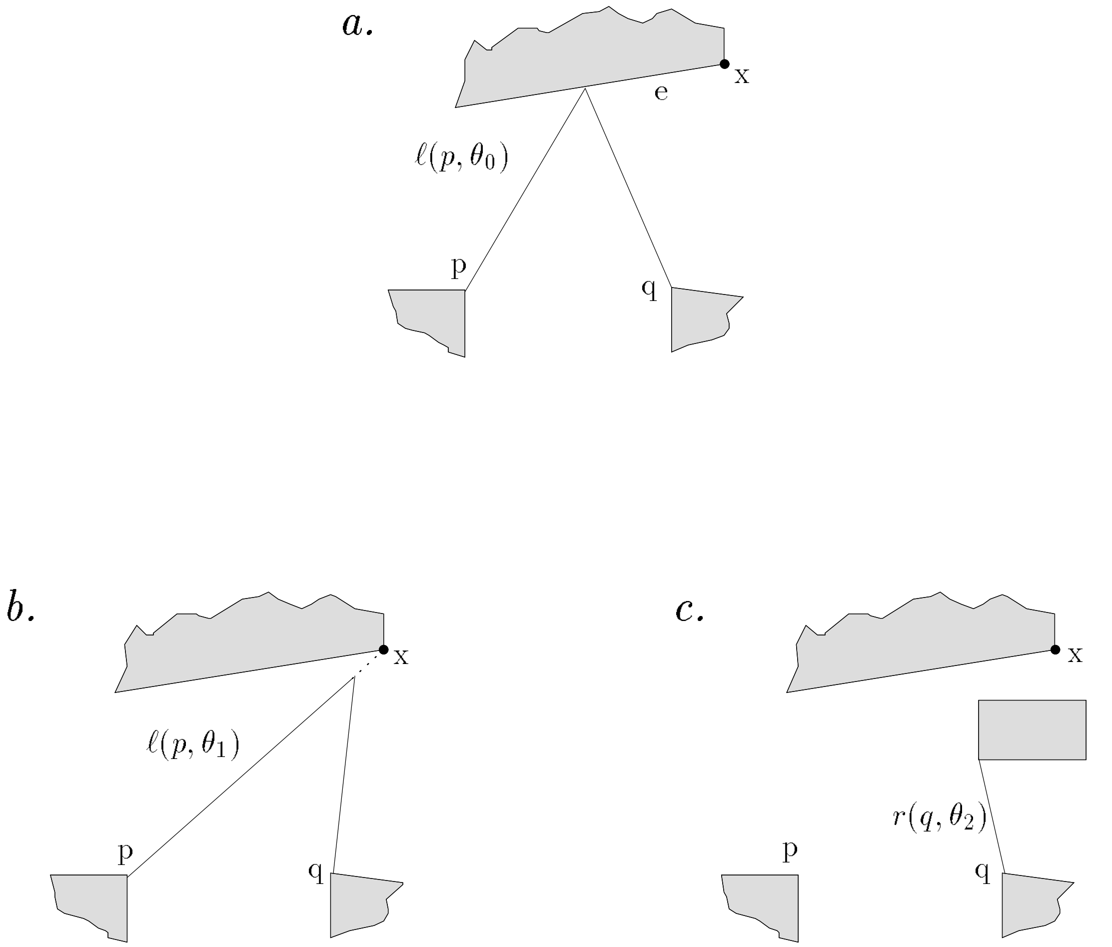

Figure 2.5: Vertex critical event is charged to a visibility edge. Figure a shows the vertex event. Figures b and c show the previous vgraph events to which the vertex event may be charged.

2.3 Topological changes to the non-directional backprojection

As θ changes, the topology of the backprojection changes when edges are inserted or deleted at critical events. Using amortization techniques, we show that over the entire range of θ, the number of topological changes to the boundary of the backprojection is bounded by $O(n^2)$. Thus we can compute a representation for the non-directional backprojection incrementally instead of computing a slice for each critical θ. Namely, we fix $\theta_0 = 0$ and compute the ordered set

$$\{B_{\theta_0}(G), \Delta B_{\theta_1}(G), \ldots, \Delta B_{\theta_\omega}(G)\}$$

where $\theta_i \in \{\theta | \theta \text{ is critical}\}$, $\theta_0 < \theta_1 < \cdots < \theta_\omega$, and $\Delta B_{\theta_i}(G)$ encodes the net change to the data structure representing backprojection slice $B_{\theta_{i-1}}(G)$ to obtain the data structure representing backprojection slice $B_{\theta_i}(G)$ by adding and deleting vertices. We show below that our representation for $B(G)$ has size $O(n^2)$ and is computed in time $O(n^2 \log n)$.

2.3.1 Complexity of the non-directional backprojection

In this section we bound the size of the non-directional backprojection by bounding the total number of changes to the backprojection over all θ.

Recall that under the assumption of general position, at most one critical event will occur at motion direction θ. Under this assumption, however, it is not true that a critical event causes only one topological change or only constant topological change to the backprojection. Indeed, we can construct examples where $O(n)$ changes occur at some θ, or where $O(n^{3/2})$ critical events occur, each causing $O(\sqrt{n})$ changes to the backprojection (see Figures 2.6 and 2.7). But in the following discussion, we show that the total number of changes to the backprojection over the full range of θ is $O(n^2)$.

In what follows, we make use of an object called a *conforming ray*. A conforming ray is specified by an anchor vertex q and is defined for the interval $[\theta_1, \theta_2]$ in which q is a sticking vertex. Crucial to this definition is the assumption that every polygonal obstacle in the environment contains at least one sticking vertex. Consider an obstacle in the configuration space. Even in the case of no friction, a motion in direction θ could stick on a vertex if θ points into the span of the normal vectors on the edges adjacent to the vertex. The sum of the angles spanning the normal vectors over all vertices is 2π, so any θ will point into the cone spanning the normal vectors at some vertex. Any vertex at which this happens could cause sticking, and so will be called a sticking vertex. See Figure 2.8 for an illustration. For each sticking vertex, we construct a *leading* conforming ray and a *trailing* conforming ray.

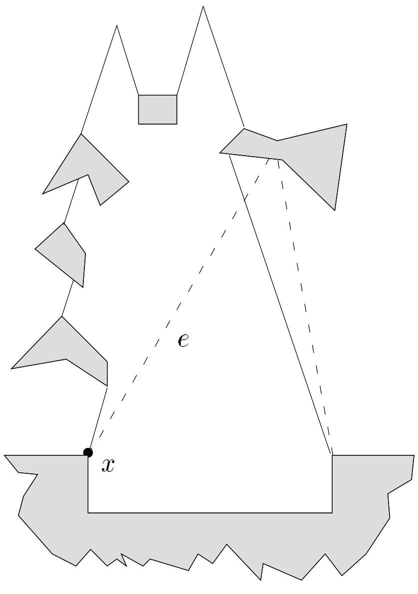

Figure 2.6: Combinatorially large change in backprojection caused by vgraph event at vertex x.

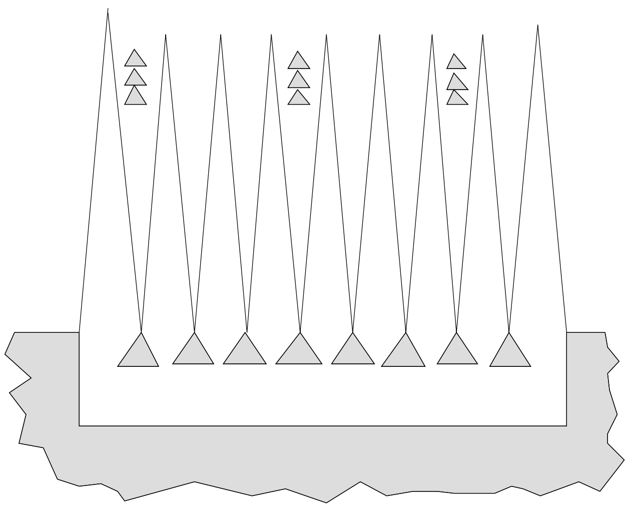

Figure 2.7: Each of the $O(n)$ spikes of the backprojection will be vgraph critical with each of the $O(\sqrt{n})$ clusters of obstacles. Each cluster is of size $O(\sqrt{n})$, so each of $O(n^{3/2})$ vgraph critical events causes $O(\sqrt{n})$ changes to the topology of the backprojection.

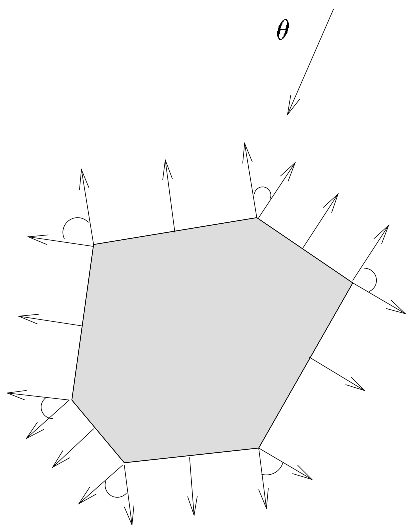

Figure 2.8: At least one vertex of the polygon must be a sticking vertex.

Definition 2.2 *A* leading conforming ray *on sticking vertex q for commanded motion θ is formed by extending $\ell(q,\theta)$ at angle $\theta+\pi-\epsilon_c$ until it intersects an obstacle. We then follow the boundary of the obstacle anticlockwise until reaching the first vertex q' that is a sticking vertex for direction θ. Treat this vertex as q and continue. A* trailing conforming ray *on sticking vertex p for commanded motion θ is formed by extending $r(p,\theta)$ at angle $\theta+\pi+\epsilon_c$ until it intersects an obstacle. We then follow the boundary of the obstacle clockwise until reaching the first vertex p' that is a sticking vertex for direction θ. Again, treat this vertex as p and continue (see Figure 2.9).*

For purposes of discussion, we say that a point in configuration space is *on* a conforming ray or *contained* in it if the conforming ray passes through the point. A point x *leaves* a conforming ray if the ray no longer passes through x. We imagine that conforming rays sweep out area continuously between critical events. If obstacle vertex x enters the boundary of the backprojection as some conforming ray sweeps over x, we say that the conforming ray *brings x into the boundary of the backprojection.* We are interested in whether a vertex is part of the *boundary* of the backprojection since the backprojection itself is neither open nor closed. For example, in the third backprojection shown in Figure 2.10, vertex z is contained in the backprojection, and vertex x is not, while both are on the boundary of the backprojection.

Note that the conforming rays bounding the backprojection move monotonically with θ in the absence of sliding events. By this we mean that between sliding events, conforming rays sweep out area in one direction as θ increases. In particular, once a point on a conforming ray leaves the conforming ray, it will not

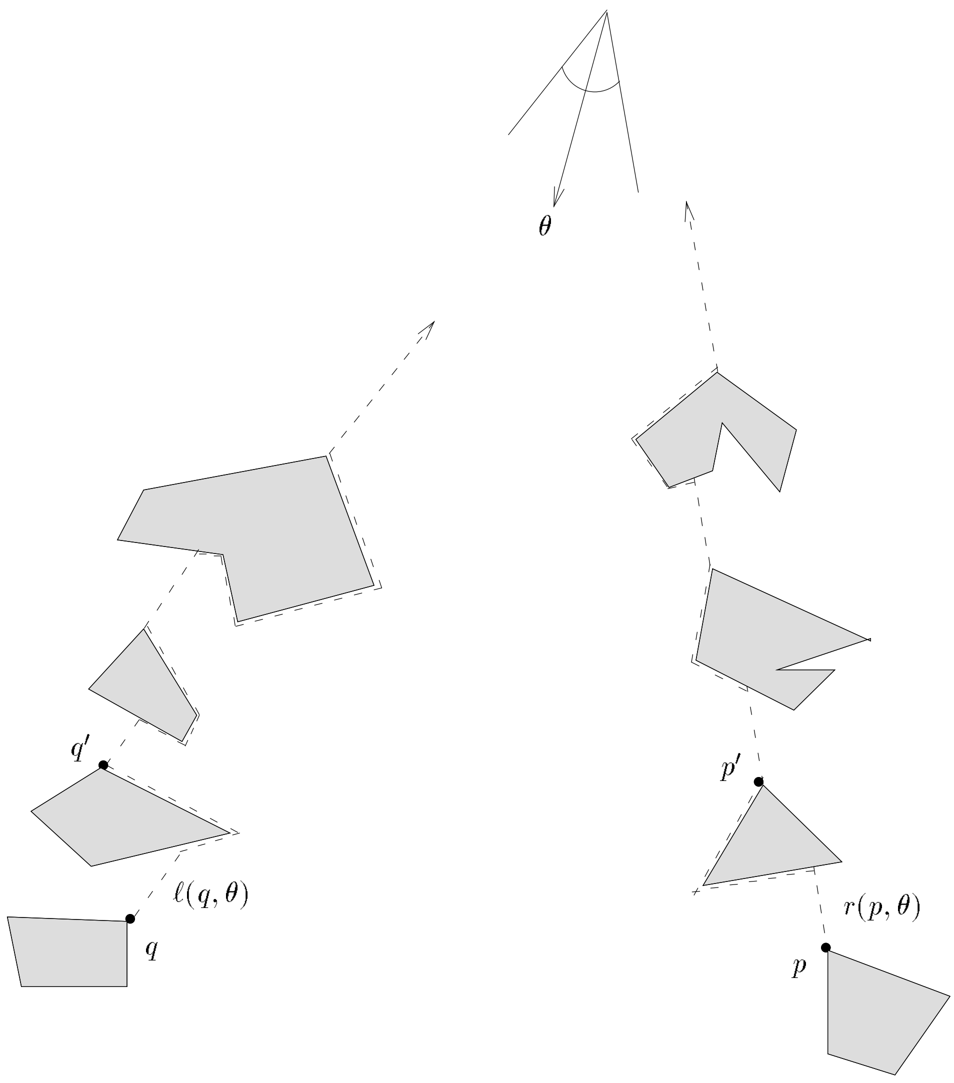

Figure 2.9: Leading conforming ray on q and trailing conforming ray on p.

become part of that conforming ray again. This is because each conforming ray is composed simply of obstacle edges and free space edges, which are parallel to the bounding edges of the control uncertainty cone and change monotonically. As critical events occur, the backprojection changes and vertices are added to or deleted from the backprojection boundary. Since a vertex is deleted only as many times as it is added, to establish a bound on the number of topological changes due to vgraph events, we need only consider insertions of vertices to the boundary of the backprojection. When x enters the boundary of the backprojection due to a vgraph event, at least one conforming ray will contain x. We can then say that x enters the backprojection boundary as this conforming ray sweeps over x, and that the entry of x can be charged to this conforming ray. Since conforming rays are monotonic between sliding events, this conforming ray will bring x into the backprojection boundary at most once. Note that even if a conforming ray could back up due to a sliding event, we could charge any resulting topological changes to the sliding event since there are only $O(n)$ sliding events. With $O(n)$ vertices in the environment and $O(n)$ conforming rays, each capable of bringing in each vertex at most once, we have $O(n^2)$ vertices entering the boundary of the backprojection due to vgraph events, *i.e.*, we have $O(n^2)$ topological changes attributable to vgraph events.

We are now ready to prove the following:

Theorem 2.3 *There are $O(n^2)$ changes to the topology of the backprojection over all values of θ.*

Proof: With the above discussion we have shown that each vertex brought into the boundary of the backprojection due to a vgraph event can be charged to a conforming ray on which it lies (see Figure 2.10). There are $O(n)$ conforming rays, and n obstacle vertices, so this gives $O(n^2)$ changes to the topology of the backprojection over all vgraph critical events.

Sliding events contribute only $O(n)$ critical values of θ, so the total number of changes to the topology of the backprojection due to sliding events is $O(n^2)$. A vertex critical event occurs when a free space vertex coincides with an obstacle edge. Since the environment is assumed to be in general position, no new constraint rays arise and therefore the only change to the boundary of the backprojection is that an obstacle edge segment is inserted or deleted. Figure 2.11 illustrates this case. Thus the $O(E)$ vertex critical events cause only local changes to the boundary of the backprojection, so they contribute only $O(E)$ topological changes.

Thus the total number of changes to the topology of the backprojection is $O(n^2)$. □

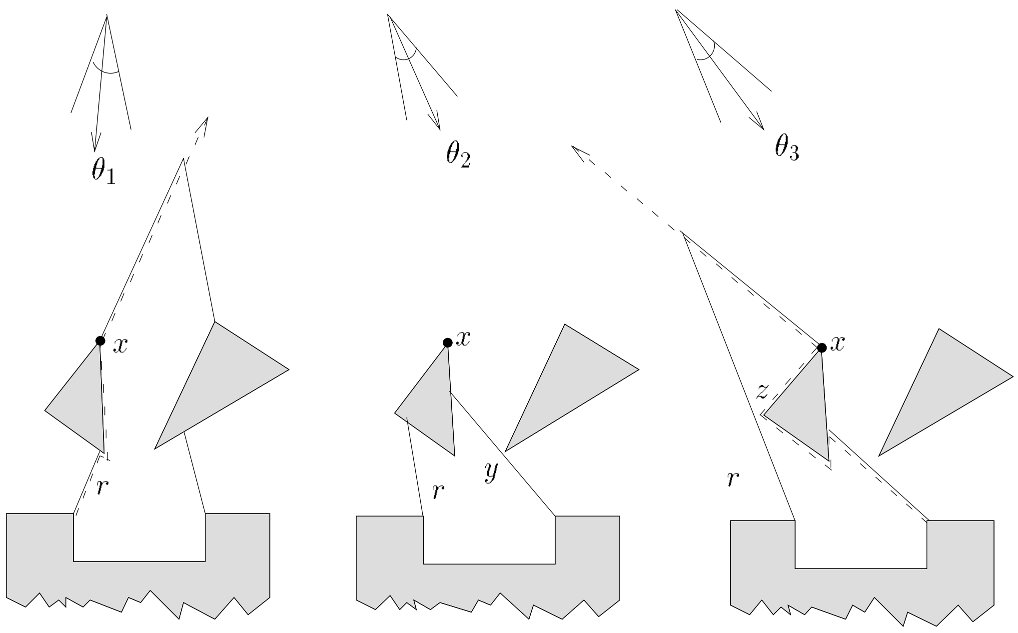

Figure 2.10: Two entries of obstacle vertex x into the boundary of the backprojection are charged to different conforming rays.

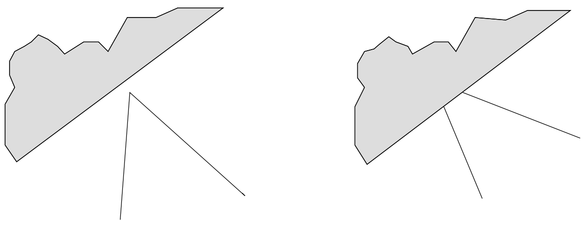

Figure 2.11: Change to boundary of backprojection due to a vertex critical event.

2.4 Algorithm One-Step

In this section we prove the following:

Theorem 2.4 *Given a goal G of constant size and an arrangement of input polygons $\mathcal{P}$ of size $O(n)$, a representation of size $O(n^2)$ for the non-directional backprojection $B(G)$ can be computed in time $O(n^2 \log n)$.*

The original arrangement of obstacles has size n and is taken as input. The visibility edges are then computed in time $O(n^2)$ [AAG+86, Wel85] where n is the number of vertices in the environment. The visibility graph and arrangement remain fixed throughout the computation, which then begins by fixing θ at an initial value, say $\theta_0 = 0$, and computing a backprojection $B_{\theta_0}(G)$. To keep the backprojection updated as θ changes, a priority queue for each type of critical value is maintained. These queues are initialized as follows:

- Sliding critical values: For each obstacle edge e, find each θ at which the determination of sliding vs. sticking on e will change. Insert the pairs (θ, e) in the sliding critical queue, ordered by θ.

- Vgraph critical values: For each constraint ray e, find each θ at which e will become coincident with a visibility edge. Insert the pairs (θ, e) in the vgraph critical queue, ordered by θ. The visibility graph, which has size $O(E)$, can be found and the critical values sorted in time $O(n^2 \log n)$.

- Vertex critical values: For purposes of this discussion, we say that free space backprojection edges e_1 and e_2 are *consecutive* if e_1 is a left edge, e_2 is a right edge, and e_2 is the next free space edge along the boundary of the backprojection. Then for each pair of consecutive free space edges on the boundary of $B_{\theta_0}(G)$, find the first θ at which their intersection point p will lie on an obstacle edge. Insert the pair (θ, p) in the vertex critical queue, ordered by θ.

Since each queue contains $O(n^2)$ elements, each requires time $O(n^2 \log n)$ to initialize and $O(\log n)$ to update.

Note that the sliding and vgraph critical values can be computed ahead of time, while the vertex critical values cannot since we do not know where the free space vertices will be. To solve this problem, we introduce a data structure to help keep track of potential vertex events. Recall that a free space edge is the maximal initial segment of a semi-infinite constraint ray. As the computation proceeds, for each constraint ray e, we keep an updated priority queue of obstacles that the ray intersects. If ray e, anchored at p, lies coincident with vertex v of obstacle j, then the edges of j incident to v are added to or deleted from the priority queue for e depending on whether e will intersect them as θ increases. Since there are

$O(n)$ constraint rays and $O(n)$ obstacle edges, the overall time to maintain these queues is $O(n^2 \log n)$. Then at each vgraph or sliding event, we find the new free space vertices and calculate their impending intersections with obstacles, based on the priority queues for their generating edges. This can be done by computing the circular arc in which the free space vertex travels and intersecting it with the common edges in the two queues. These potential vertex events are added to the vertex critical event queue, and the computation continues. As we showed earlier, there is only constant amortized change to the boundary of the backprojection for vgraph events, so there is only a constant amortized number of new free space vertices due to a single vgraph event.

Suppose we have some slice of the non-directional backprojection computed and wish to compute the next backprojection slice. The next critical value of θ can be found by comparing the first elements of the three priority queues, and removing the minimum valid event. Note that an event is valid if it arises from a vertex or edge that is part of the backprojection; not all queued events will be valid. If the next event arises from a sliding criticality, then a new backprojection slice can be computed using the plane sweep algorithm, since there are only $O(n)$ sliding critical values. In the case of a vertex critical value, the only change to the backprojection will be that a free space vertex either enters or leaves the boundary of an obstacle. The backprojection boundary can then be updated locally to reflect the change. On the other hand, a vgraph criticality can cause $O(n)$ changes to the backprojection (see Figure 2.6). Since there are potentially $O(n^2)$ vgraph critical values, we would like to update the backprojection incrementally when one arises. This can be accomplished by deleting the critical edge e from the backprojection and then tracing out the new backprojection region starting at one endpoint of e until some vertex on $B_\theta(G)$ is reached (see Figure 2.6). The following lemma shows that this can be accomplished in time linear in the size of the change.

Lemma 2.5 *Given polygonal environment $\mathcal{P}$, goal G, commanded motion direction θ, control uncertainty ϵ_c, and backprojection vertex u, vertex v adjacent to u on the backprojection $B_\theta(G)$ can be found in $O(1)$ time, making use of a data structure that costs $O(n^2 \log n)$ to maintain over the entire computation.*

Proof: For each sticking vertex v, we keep a priority queue of constraint rays that intersect the left ray of v, ordered by the distance from v to the ray intersection. Each time a new constraint ray is created or deleted at a sliding critical event, we update the priority queues. Since there are $O(n)$ rays, and each is inserted into $O(n)$ queues at a cost of $O(\log n)$ per insert, the overall time to maintain the data structure is $O(n^2 \log n)$. □

As the non-directional backprojection $B(G)$ is computed, we can check incrementally for containment of the start region. Some or all of the vertices of $R \subset \mathcal{P}$ may be in free space, so we say that a *pseudo-critical* event occurs when an edge of R intersects $B_\theta(G)$. If a *pseudo-critical* event occurs at θ_j, then we test the backprojection at $B_{\theta_j}(G)$ for containment of R. To do this, we maintain an updated backprojection slice $B_{\theta_i}(G)$ for $\theta_i \in \{\theta | \theta \text{ is critical}\}$ as the computation proceeds. The plane sweep algorithm [Don89] is used to decide $R \subset B_{\theta_j}(G)$, for $\theta_j \in \{\theta | \theta \text{ is pseudo-critical}\}$, which requires time $O(n \log n)$ for R of constant size. If θ is found such that $R \subset B_\theta(G)$, then motion direction θ from R is guaranteed to reach the goal. Otherwise, no one-step motion is guaranteed to reach G. We have thus established the following:

Theorem 2.6 *The one-step planar compliant motion planning problem with uncertainty can be solved in time $O(n^2 \log n)$.*

2.5 Discussion

In this chapter we have presented an $O(n^2 \log n)$ algorithm solving the problem of one-step compliant motion planning in the plane. This work has direct application in mechanical assembly planning, in particular for tasks involving constrained motion, such as inserting a peg in a hole. Furthermore, the path planning methods developed here could be used in a system such as the robot surveillance system described in Chapter 5.

We restricted our attention in this chapter to the question of computing goal *reachability*, *i.e.*, from where can a goal be reached. In the next chapter we begin to address the question of goal *recognizability*. In particular, we develop methods to compute the set of sensor placements from where goal attainment can be recognized.

Chapter 3

Sensor Configuration

In this chapter we begin our investigation of *task-directed sensing*, where sensing is driven by the information needs of the task to be performed. We introduce a computational framework in which to study sensor configuration, and give combinatorially precise algorithms for specific cases.

A primary motivation for this work is in the domain of cooperating robots. Suppose robot A is performing a task, and robot B is equipped with sensors and should "watch" robot A. Then our algorithms can be used, for example, to compute the regions that B should not enter, if A is to remain visible. Less generally, A and B could be part of the same robot, for example, a physically distributed but globally controlled system.

For another example of where such strategies would be useful, consider the Error Detection and Recovery Framework [Don89]. An *Error Detection and Recovery* (EDR) strategy is one that is guaranteed to reach the goal when the goal is recognizably reachable, and signals failure otherwise. Our algorithms can be used in conjunction with an EDR planner to compute where a sensor should be placed in order to recognize success or failure of a motion plan. We discuss this application in Chapter 4.

Many other applications of automatic sensor configuration arise in the area of robot surveillance. In particular, our algorithms apply to the problems of intruder detection, execution monitoring, and robot reconnaissance. A natural extension of our work can be made to the problem of beacon placement for robot navigation. In Chapter 5 we discuss a robot surveillance demonstration system that we built using an implementation of some of the algorithms in this chapter.

3.1 Goal recognizability

Let's start with the example of two cooperating mobile robots. Suppose robot A is executing a motion plan and should reach a goal G. Let B be a robot equipped with a sensing device. Where should robot B move if it wants to detect when robot

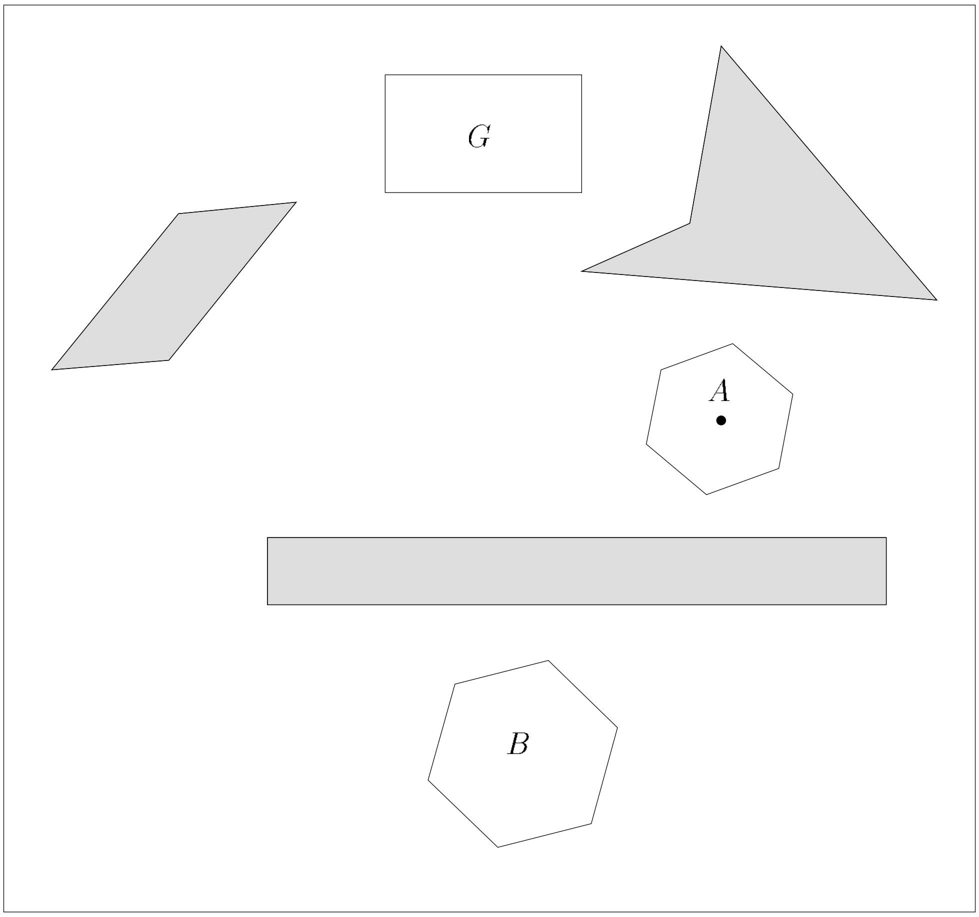

Figure 3.1: A sensor configuration example. Robot B must be configured so that it can detect when robot A has reached goal G. Robot A is said to be in the goal when its reference point (here, the black dot) lies in G.

A has reached the goal? We assume that we have a map of the environment, which we use to determine feasible sensor placements. The map consists of polygonal obstacles, which may include a bounding polygon representing the boundary of the environment. See Figure 3.1 for an illustration.

Although the robots exist and move in a three-dimensional world, they always maintain contact with the floor, so we can simplify the planning problem by projecting the robots and their environment onto the plane. Then we can ask whether or not the two dimensional "footprint," or projection of robot A onto the plane, is visible from a given sensor location in the plane. Now we have transformed the problem of detecting a three-dimensional robot into a planar visibility problem. We say that our environment is a $2\frac{1}{2}D$ geometric environment because the obstacles are considered to be swept polygons.

We now make these notions precise. Our robot configuration space is denoted C_r, and in this chapter we consider two types of planar robot motion. We have $C_r = \mathbb{R}^2$ when the robot has two translational degrees of freedom and a fixed orientation. When the robot is allowed to both rotate and translate in the plane, the robot configuration space is $\mathbb{R}^2 \times S^1$.

The sensor we employ is an idealized but physically realizable model of a point-and-shoot sensor, such as a laser range finder. A sensor configuration is specified by a *placement* and a viewing direction, or *aim*. When in a particular configuration, the sensor returns a distance reading to the nearest obstacle, which is accurate to within some known bound. Such a ranging device has been developed in our robotics laboratory at Cornell, and has been used for both map-making and robot localization. Details of how the sensor operates can be found in the work of Brown, Chew, and Donald [Bro95,BCD93].

We denote the space of sensor placements $C_{s_p} = \mathbb{R}^2$ and the space of sensor aims $C_{s_c} = S^1$. Our sensor configuration space is $C_s = C_{s_p} \times C_{s_c} = \mathbb{R}^2 \times S^1$. For a given sensor configuration $(\mathbf{p}, \psi)$, the sensor returns distance readings for a subset of $\mathbb{R}^2$. We call this subset the *sensitive volume* of the sensor, and denote it by $SV(\mathbf{p}, \psi)$. Figure 3.2 illustrates an example of the sensitive volume for a general sensor at position $\mathbf{p} \in C_{s_p}$ and pointed in direction $\psi \in C_{s_c}$. In what follows, we restrict our attention to questions of visibility within the sensitive volume.

Definition 3.1 *Given a subset X of the robot configuration space C_r, the* recognizability region *of the robot in X is the set of all sensor configurations from which the sensor can detect the robot when its reference point lies in X.*

In Sections 3.3 and 3.4, we give algorithms for computing the recognizability region of a robot in a goal under two different notions of visibility. We justify our use of visibility as a model for detectability by noting that the recognizability region of a robot given most sensors is a subset of the region from which the robot is visible. For an idealized ranging device, the visibility region of the robot is equivalent to the recognizability region. We show in Section 3.5 how the error characteristics of an imperfect sensor can be used to construct the recognizability region from the visibility region.

3.2 Visibility as a model for detectability

The main algorithmic paradigm we use in the computation of recognizability regions is polygon visibility, as defined in Section 1.1.6. Polygon visibility questions in the plane typically involve the computation of regions inside a polygon that are visible from some point or edge inside the polygon. For example, Figure 3.3 shows the region completely visible from a point inside a polygon, and Figure 3.4 shows the region weakly visible from an edge inside the polygon. Here, we extend the relevant definitions from the internal case to define *external* polygon visibility.

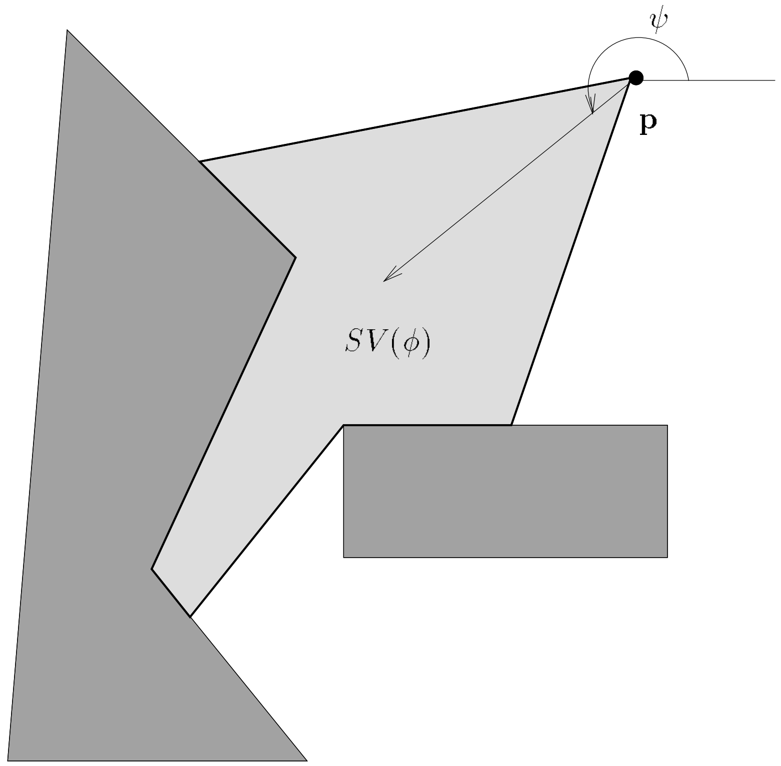

Figure 3.2: An example sensitive volume for a general sensor at configuration $\phi = (\mathbf{p}, \psi)$. The sensor is placed at position $\mathbf{p}$ and pointed in direction ψ. The sensitive volume is the lightly shaded region $SV(\phi)$. It is partly bounded by obstacles (darkly shaded).

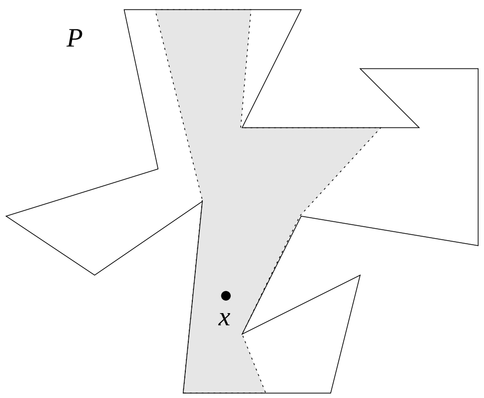

Figure 3.3: An illustration of complete visibility inside a polygon. The shaded region inside polygon P is the region completely visible from point x.

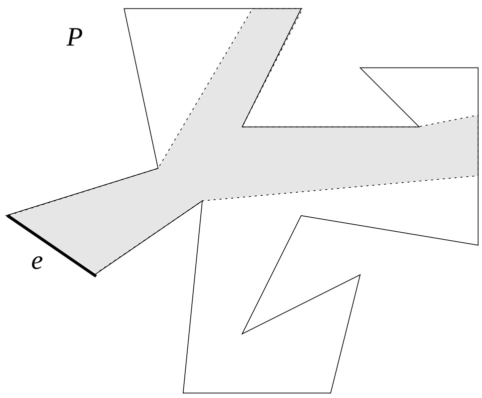

Figure 3.4: An illustration of weak visibility inside a polygon. The shaded region inside polygon P is the region weakly visible from edge e.

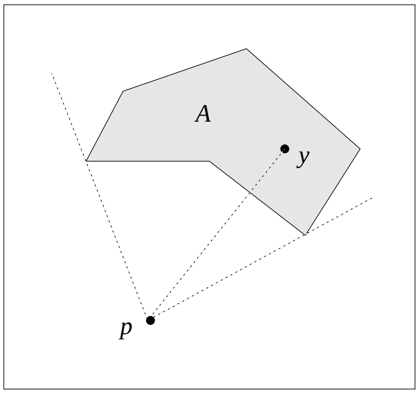

Figure 3.5: Polygon A is completely visible from point p. For no point y in A does the line segment $\overline{py}$ intersect an obstacle.

Definition 3.2 *We say that polygon A is* completely visible *or* unobstructed *from a point p external to A if every point y in the closure of A is visible from p, where A itself is not considered an obstacle.*

That is, A is completely visible from p if for no point y in A or on the boundary of A does the segment $\overline{py}$ intersect an obstacle segment (except possibly at the endpoints). See Figures 3.5 and 3.6 for illustrations.

Definition 3.3 *Polygon A is* weakly visible *from p external to A if at least one point in the closure of A is visible from p.*

See Figure 3.7 for an illustration.

Detecting whether or not a polygon is weakly externally visible from a given point can be answered in $O(n)$ time [BKT89]. We give algorithms for explicitly constructing the locus of points from which a polygon is visible, and we analyze how this visibility region changes as the distinguished polygon moves.

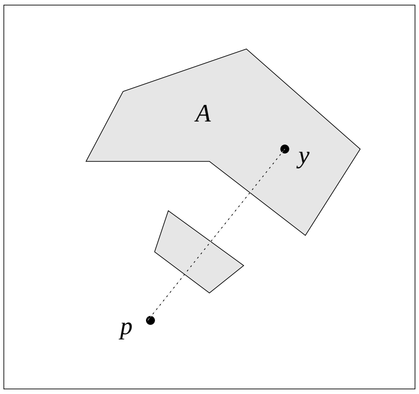

Figure 3.6: Polygon A is not completely visible from point p. There exists at least one point y in A such that the line segment $\overline{py}$ intersects an obstacle.

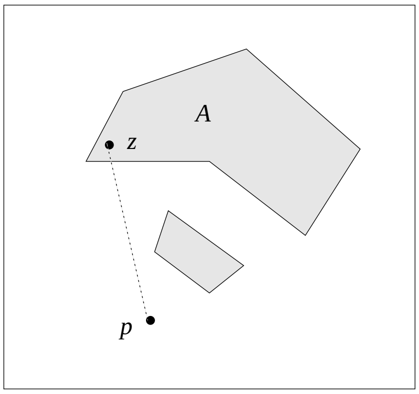

Figure 3.7: Polygon A is weakly visible from point p. There exists at least one point z in A such that the line segment $\overline{pz}$ does not intersect an obstacle.

3.3 Complete visibility

In this section we consider a simplified version of the robot detection problem, in which the computed sensor placements are those that allow an unobstructed view of the robot. We give algorithms for detecting a stationary robot, and for detecting a robot at any position and orientation within a goal region. The computation of placements affording an unobstructed view is closely related to the notion from computational geometry of complete polygon visibility, as defined in Section 1.1.6. We show in Section 3.5 how the visibility regions can be restricted to account for the error characteristics of the sensor.

Our result is the following:

Theorem 3.1 *The recognizability region of a robot translating and rotating through a goal with k vertices can be computed in time $O(n\alpha(n) + nk)$ in an environment with n vertices in the complete visibility model.*

3.3.1 The complete visibility algorithm for a stationary robot

We say that robot A at configuration $\mathbf{q} = (x, y, \theta)$ is *completely visible* from sensor placement $\mathbf{p} \in C_{sp}$ if for no point y on the boundary of the robot does the segment $\overline{\mathbf{p}y}$ intersect an obstacle. Note that $\overline{\mathbf{p}y}$ may intersect $A_{\mathbf{q}}$. If $A_{\mathbf{q}}$ is completely visible from $\mathbf{p}$ in the presence of the obstacles then we say that the sensor at $\mathbf{p}$ has an *unobstructed view* of the robot at configuration $\mathbf{q}$. Our algorithm assumes that no obstacle lies within the convex hull of $A_{\mathbf{q}}$, *i.e.*,

$$\bigcup_i (CH(A_{\mathbf{q}}) \cap B_i) = \emptyset.$$

For example, an obstacle placement such as the one depicted in Figure 3.8 is not allowed.

The idea is that each obstacle casts *shadows* with respect to the robot. Each shadow is a subset of the sensor placement space C_{sp} from which the robot is partially occluded. See Figure 3.9 for an illustration. To compute the set of placements from which the robot at configuration (x, y, θ) is completely visible, we use the following algorithm:

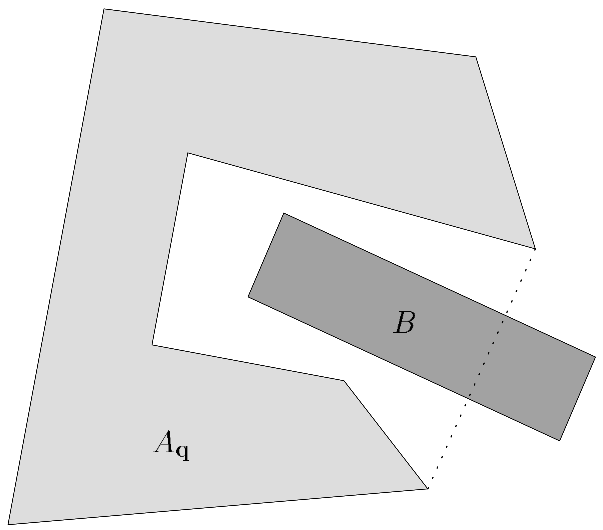

Figure 3.8: A robot configuration not considered by the algorithm. Obstacle B lies within the convex hull of robot A.

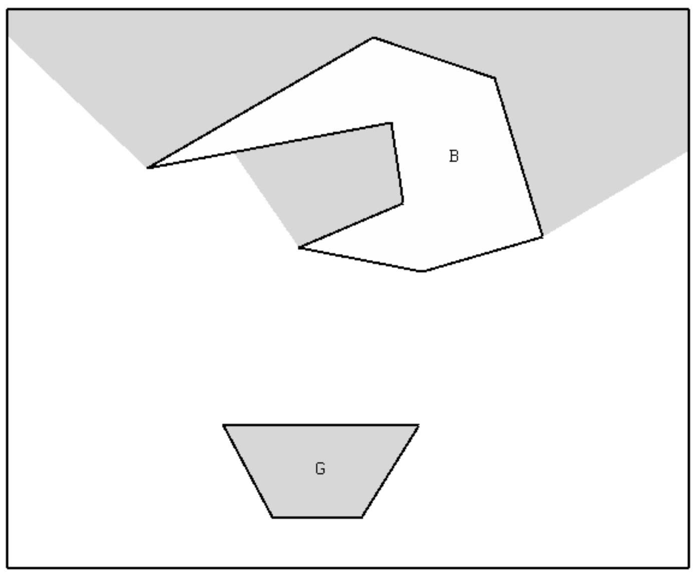

Figure 3.9: Shadows cast by obstacle B with respect to goal G in the complete visibility model. The shadows regions are shown darkly shaded.

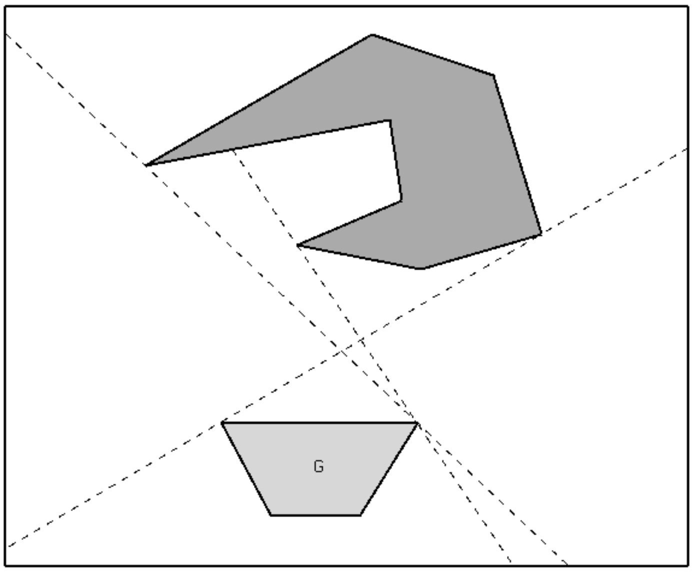

Figure 3.10: The local inner tangents between the goal and an obstacle are shown dashed.

Complete visibility algorithm for a stationary robot

1. Construct all local inner tangents between the obstacles and the goal. Figure 3.10 gives an illustration. Represent each tangent as a ray anchored on a vertex of the goal. See Figure 3.11 for an illustration.

2. Extend each tangent ray starting at the point of tangency with an obstacle until it hits an edge of the environment (an obstacle or the bounding polygon). We call these segments *visibility rays*. See Figure 3.12 for an illustration.

3. Consider the arrangement of all the polygons in the environment, along with these visibility rays. Compute the single arrangement cell that contains the goal polygon. Figure 3.13 gives an illustration.

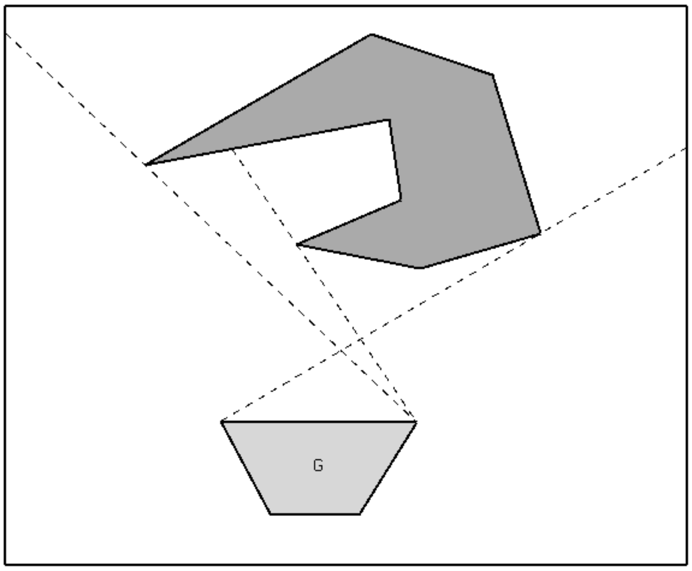

Figure 3.11: The tangent rays between the goal and an obstacle are shown dashed.

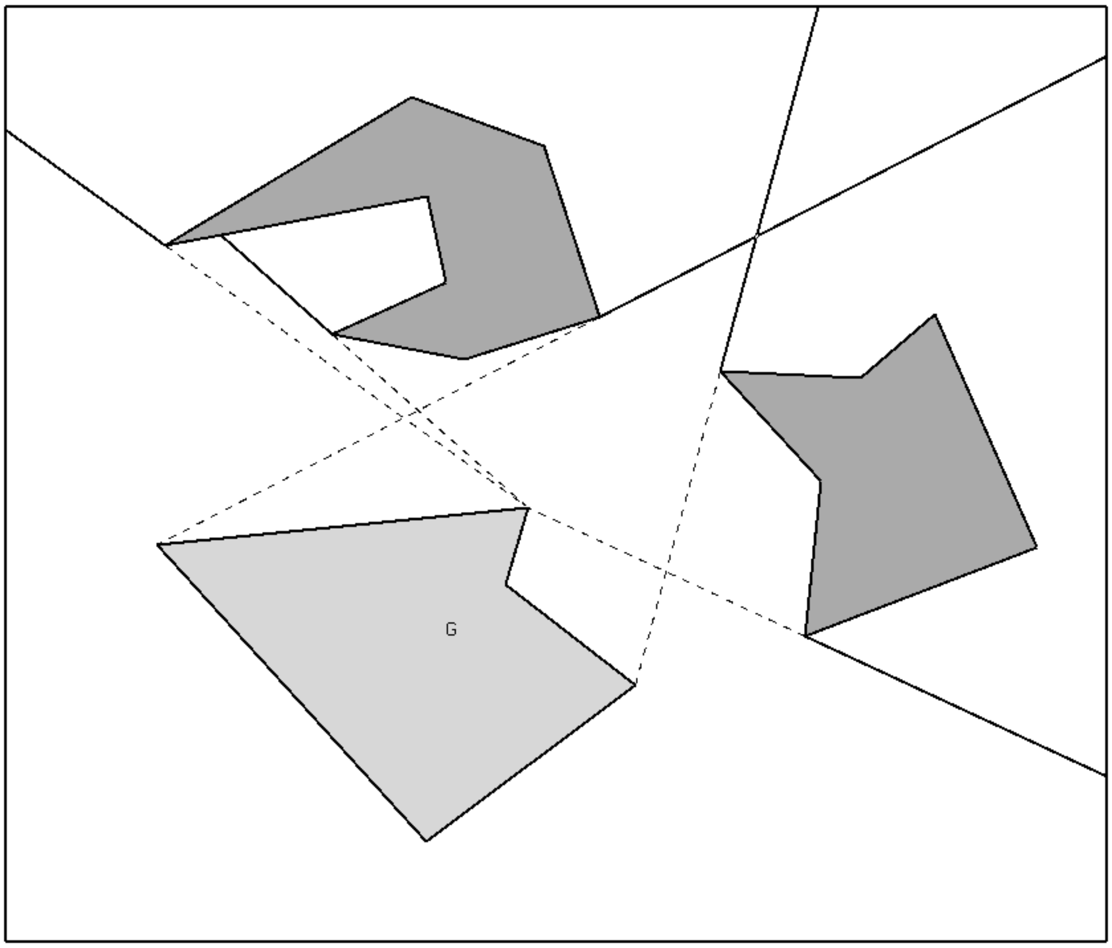

Figure 3.12: The segments of the tangent rays used in the arrangement computation are shown solid.

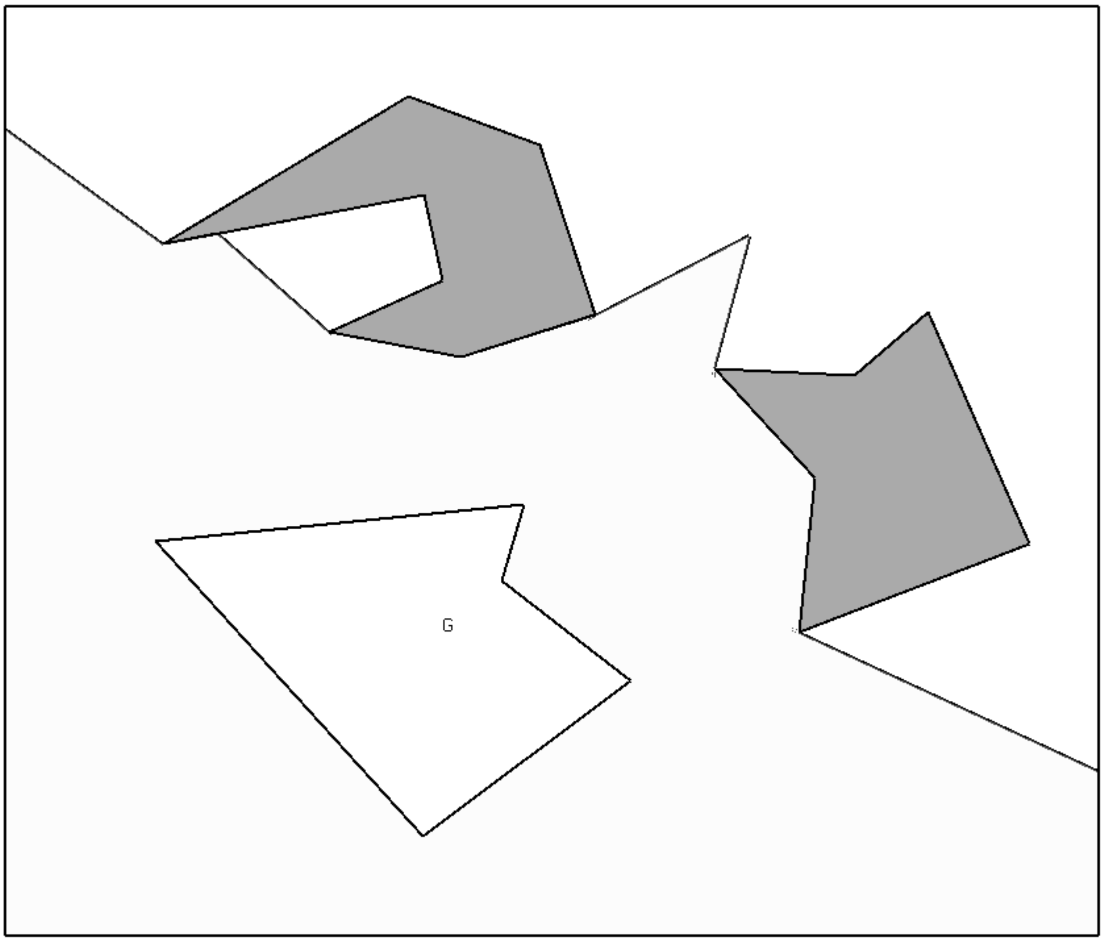

Figure 3.13: The arrangement cell containing G is shown lightly shaded.

3.3.2 Complexity of the complete visibility algorithm

Let A be a polygon representing the robot, and B_i be a polygonal obstacle. If A has m vertices and obstacle B_i has n_i vertices, we can compute the $O(n_i)$ local inner tangents between A and B_i in time $O(mn_i)$. For an environment of obstacles with n vertices overall, we can compute the $O(n)$ local inner tangents in time $O(mn)$. Computing a single cell in an arrangement is equivalent to computing the lower envelope of a set of line segments in the plane, which for a set of size n takes time $O(n\alpha(n))$, where $\alpha(n)$ is the inverse Ackerman function [PSS86]. Thus, the overall time for computing the recognizability region of a stationary polygon in the complete visibility model (as in Definition 3.2) is $O(n\alpha(n) + mn)$.

3.3.3 Complete visibility over a region

We now consider sensor placements with an unobstructed view of the robot at any position or orientation within a goal region G. We model the robot as a connected polygon, with a reference point that lies inside the polygon. Note that the robot is said to "lie in the goal" if and only if its reference point lies in the goal. The idea is that a sensor placement is valid if and only if, as its reference point moves within the goal, the entire swept area covered by the robot is visible. We present two algorithms below.

Complete visibility algorithm for a translating robot

First consider the case where the robot has a fixed, known orientation. We further restrict the robot's motion to pure translation. Denote the robot at orientation θ by A_θ. Consider the Minkowski sum $A_\theta \oplus G$. See Figure 3.14 for an illustration. $A_\theta \oplus G$ is the set of points A_θ can occupy when the reference point lies in G. The complete visibility region for the polygon $A_\theta \oplus G$ is the set of all sensor placements from which A_θ can be completely seen when its reference point lies anywhere in G.

To compute the shadow boundaries, we introduce local inner tangents between each obstacle and the convex hull of $A_\theta \oplus G$, denoted $CH(A_\theta \oplus G)$. Note that this is not an approximation; only the outermost tangents with the distinguished polygon (in this case $A_\theta \oplus G$) generate shadows in the complete visibility model.

We can compute the convex hull of $A_\theta \oplus G$ efficiently by exploiting the fact that for polygons A and B [Loz83]

$$CH(A) \oplus CH(B) = CH(A \oplus B).$$

Note that A and B do not need to be convex. So instead of computing $CH(A_\theta \oplus G)$ explicitly, we simply convolve $CH(A_\theta)$ and $CH(G)$.

If inner tangent e is locally tangent at obstacle vertex v, then we again introduce a visibility ray that extends e away from vertex v. The arrangement of the visibility rays and the environment now partitions C_{sp} into shadow regions and visibility

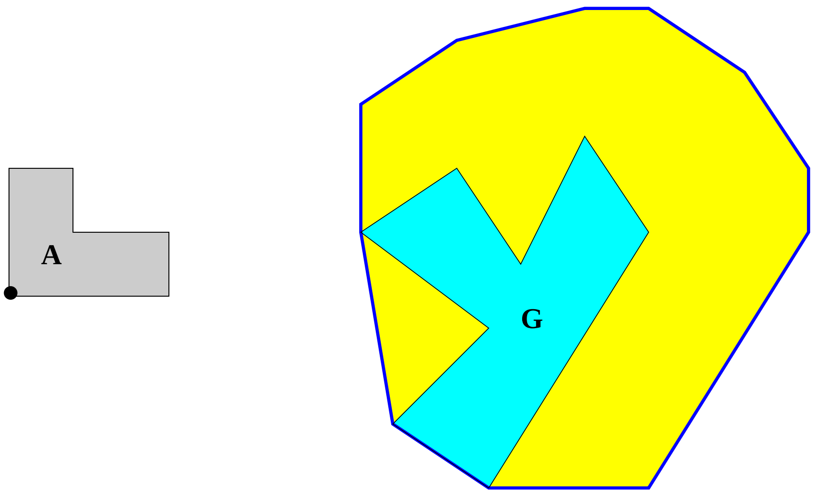

Figure 3.14: A represents the robot at a particular orientation θ; the pentagon represents the goal G; the region around G is the convolution $A_\theta \oplus G$.

regions, *i.e.*, regions from which $A_\theta \oplus G$ is partially occluded or entirely visible. But instead of computing the entire arrangement, we again note that it suffices to compute a single cell in the arrangement, namely the cell containing the goal.

Complete visibility algorithm for a translating and rotating robot

Now consider the case of a robot rotating and translating through the goal. We want the set of placements from which no portion of the robot's boundary is occluded by obstacles no matter what position or orientation the robot has within the goal. We take the longest Euclidean distance from the robot's reference point to a vertex of the robot. We call this distance the *radius* of the robot. Suppose the robot has a radius of r and its reference point lies at $\mathbf{w}$. Then the disc of radius r centered at $\mathbf{w}$ is equivalent to the area covered by the robot as it rotates around its reference point.

Hence, for robot A with radius r, the Minkowski sum of the goal G with the disc of radius r represents the swept area covered by the robot as it translates and rotates through the goal. Call this Minkowski sum the *swept goal region* $M(G, A)$. See Figure 3.15 for an illustration. We now compute the set of sensor positions that have an unobstructed view of $M(G, A)$.

To compute the shadow boundaries, we introduce local inner tangents between each obstacle and the convex hull of $M(G, A)$. This can be accomplished by simply computing all inner tangents between each obstacle and the disc of radius r at each vertex of G, then taking the outermost tangents at each obstacle. Once we have these inner tangents, the rest of the algorithm is the same as above.

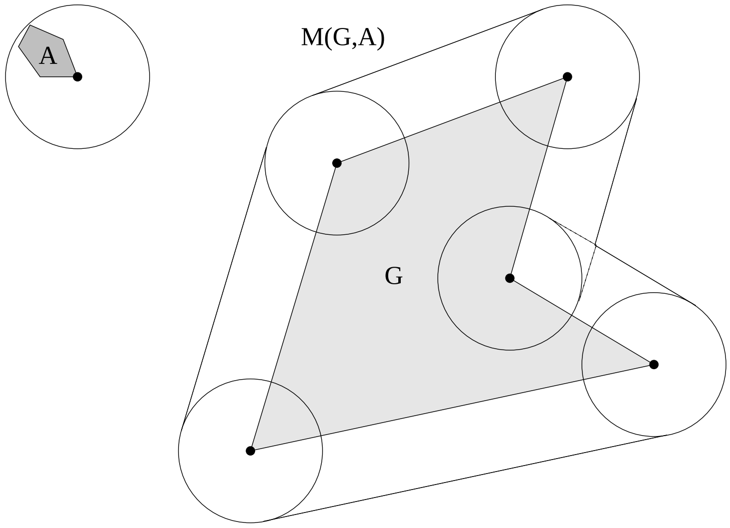

Figure 3.15: Robot A, goal G, and the swept goal region $M(G, A)$.

3.3.4 Complexity of the complete visibility algorithm over a region

In the translation-only case, we first compute the convex hull of $A_\theta \oplus G$. To do this efficiently, we use the fact that $CH(A_\theta \oplus G) = CH(A_\theta) \oplus CH(G)$ [Loz83]. Note that this does not assume A or G are convex. If A has m vertices and G has k vertices, $CH(A_\theta)$ and $CH(G)$ can be computed in $O(m \log m)$ and $O(k \log k)$ time, respectively [Gra72,Jar73,GY83]. As we saw in Chapter 1, the convolution of two convex polygons can be computed in linear time in the total number of vertices and has linear size. So $CH(A_\theta \oplus G)$ has complexity $O(m + k)$ and can be computed in time $O(m \log m + k \log k)$.

Computing the $O(n)$ local inner tangents between $CH(A_\theta \oplus G)$ and the environment can be done in time $O(n(m + k))$. The complete visibility region is the arrangement cell containing $CH(A_\theta \oplus G)$. As mentioned above, computing a single cell in an arrangement is equivalent to computing the lower envelope of a set of line segments in the plane. So the overall time to compute the visibility regions for a robot translating through the goal is $O(n\alpha(n) + n(m + k))$.

In the case of a robot rotating and translating through the goal, the only difference between the algorithm given here and the one given in Section 3.3.1 for a stationary robot is that instead of computing tangents between a stationary robot and the obstacles, we convolve a disc with the goal and compute tangents between the result of this convolution and the obstacles. In terms of complexity, the algorithms differ only in that the goal complexity rather than the robot complexity is relevant. Assuming that we know the robot radius r, we can compute $M(G, A)$ for a goal G of size k in time $O(k)$. If obstacle B_i has n_i vertices, we can compute the $O(n_i)$ local inner tangents between B_i and the convex hull of $M(G, A)$ in time

$O(kn_i)$. For an environment of obstacles with n vertices overall, we can compute the $O(n)$ local inner tangents in time $O(kn)$. So the overall time for computing the recognizability region for a robot rotating and translating through the goal in the complete visibility model is $O(n\alpha(n) + nk)$.

3.4 Weak visibility

We turn now to the question of computing recognizability regions in the *weak visibility* model (as in Definition 3.3). First we consider the problem of detecting a stationary robot within a polygonal obstacle environment. We then apply these tools to the problem of detecting a translating robot as it enters the goal.

We will show the following:

Theorem 3.2 *In the weak visibility model, the recognizability region of a robot translating through a goal of size k can be computed in time $O(kmn^3(n+m))$ for an environment of complexity n and a robot of complexity m.*

3.4.1 The weak visibility algorithm for a stationary robot

For robot A at configuration $\mathbf{q} \in C_r$, we construct the weak visibility region using an approach similar to that given by Suri and O'Rourke for computing the region weakly visible from an edge [SO86]. (See Figure 3.4 for an illustration of the region weakly visible from an edge inside a polygon.) Our algorithm is as follows:

Weak visibility algorithm for a stationary robot

1. Construct the visibility graph for the entire environment, consisting of distinguished polygon A and obstacles B. See Figure 3.16 for an illustration.

2. Extend each edge of the visibility graph maximally until both ends touch an edge of the environment. If neither of the endpoints of the extended visibility edge lie on the polygon A, discard the visibility edge. Otherwise, clip the edge at its intersection with A and call this piece a *visibility ray*. See Figure 3.17 for an illustration.

3. For each vertex v in the environment, perform an angular sweep of the visibility rays incident to v. If A remains visible to v throughout the swept angle between two adjacent visibility rays anchored at v, then the triangular swept region is output as a *visibility triangle.* Figure 3.18 gives an illustration.

The union of these visibility triangles forms the region from which A is weakly visible. The complement of the union of triangles and the environment is a collection of holes in the visibility region, which we call *shadows.* Figure 3.19 shows the

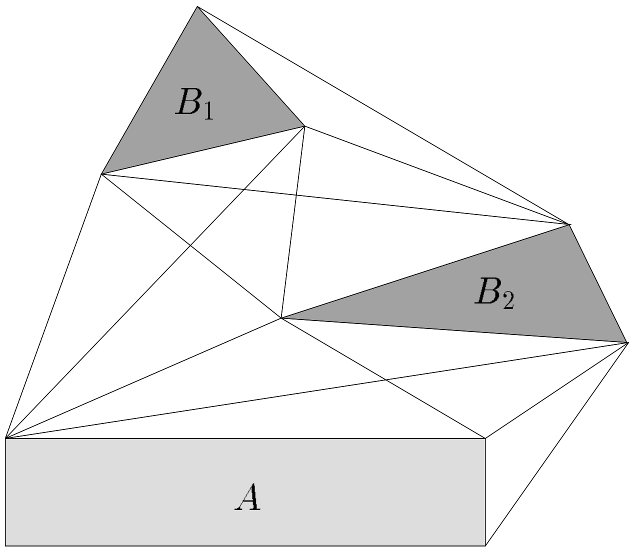

Figure 3.16: The visibility graph between robot A and obstacles B_1 and B_2.

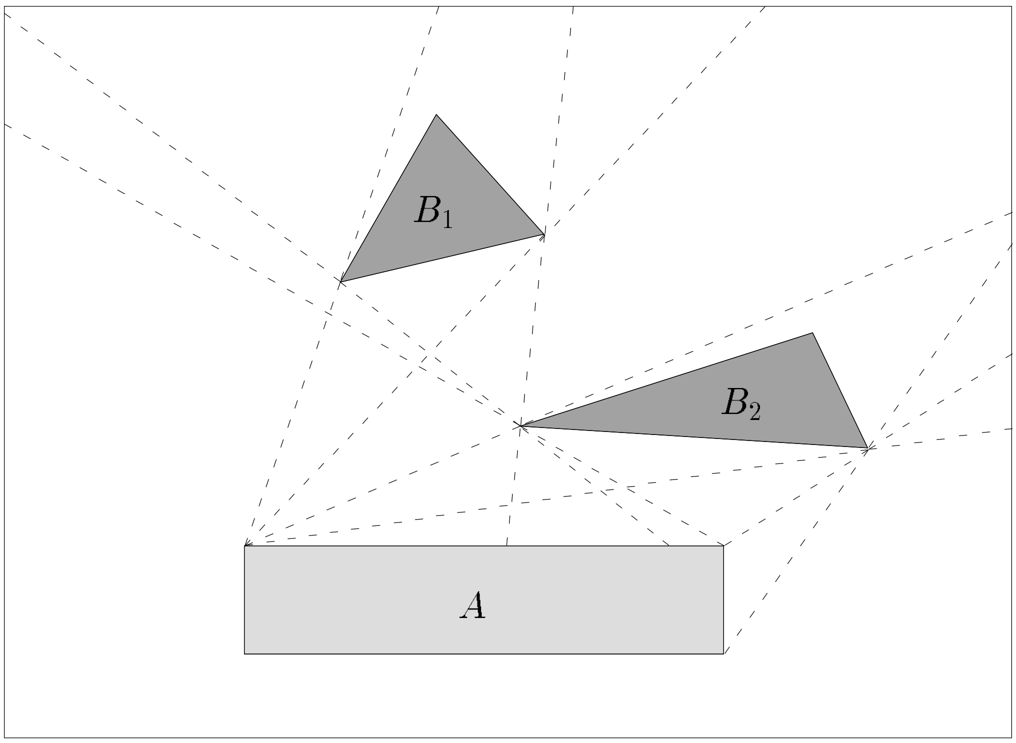

Figure 3.17: The visibility rays between robot A and obstacles B_1 and B_2 are shown dashed.

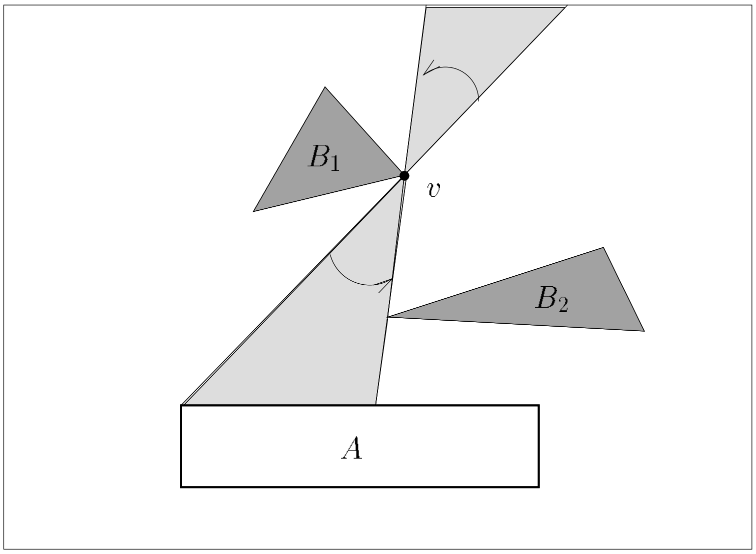

Figure 3.18: An angular sweep between two visibility rays at vertex v. The lightly shaded regions are visibility triangles.

shadows for an example environment. This example demonstrates the fact that in the partial visibility model, shadows are not necessarily bounded by tangents between an obstacle and the goal. For details on how the visibility triangles are computed, see Section 5.1.1.

3.4.2 Complexity of the weak visibility algorithm

Suppose the obstacles and bounding polygon together have n vertices, and the robot has m vertices. The visibility graph for this environment, the basic data structure used in the algorithm, has size $O(n(n+m))$. Note that we are not interested in visibility edges between the vertices of the robot itself. The extended visibility graph will, in practice, have fewer edges than the basic visibility graph, since we only keep the edges whose extensions intersect the robot. Its worst-case complexity, however, remains $O(n(n+m))$. Each vertex of the environment has $O(n+m)$ visibility rays incident to it. Therefore each vertex contributes $O(n+m)$ visibility triangles, so we have $O(n(n+m))$ visibility triangles overall. In general, the union of these triangles has complexity $O(n^2(n+m)^2)$. As was mentioned in the paper by Suri and O'Rourke [SO86], the triangles can be output in constant time per triangle: Asano *et al.* have shown that the visibility edges at a vertex v can be obtained sorted by slope in linear time with Welzl's algorithm for computing the visibility graph [Wel85,AAG+86]. Thus, the overall time for explicitly computing the boundary of the weak visibility region for robot A at any fixed configuration $\mathbf{q}$ is $O(n^2(n+m)^2)$. The region can be given as a union of triangles, without

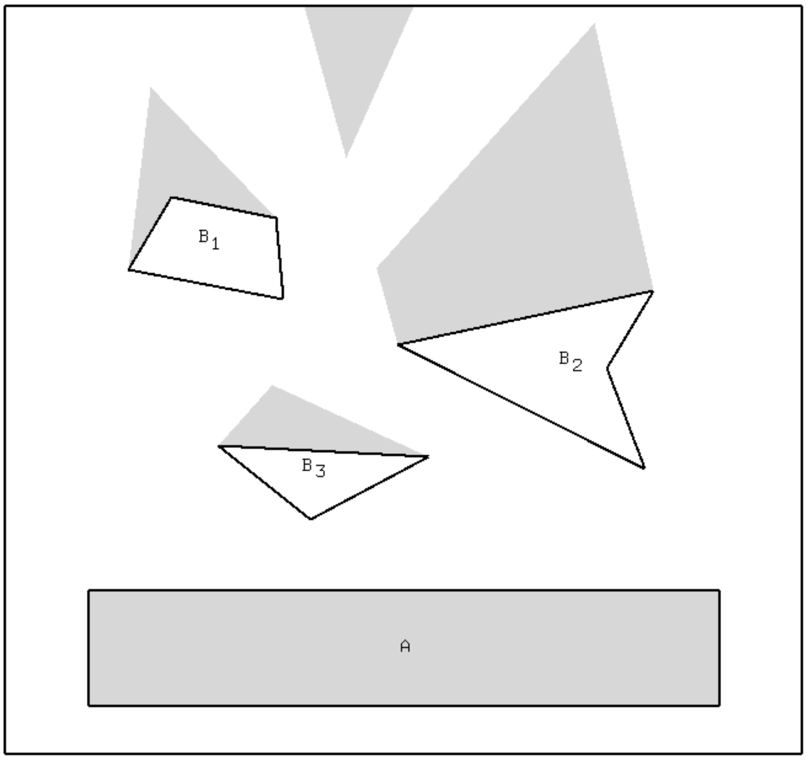

Figure 3.19: The shadows cast by obstacles B_1, B_2, and B_3 are shown shaded. The complement of the shadows, the obstacles, and the robot forms the weak visibility region of robot A.

computing the boundary, in time $O(n(n+m))$.

3.4.3 Weak visibility over a region

The algorithm above solves the problem of detecting a stationary robot in the weak visibility model. We now address the problem of maintaining line-of-sight contact with the robot as it moves within the confines of a particular polygon, for example, as the robot moves within the goal. How do the visibility triangles and shadows change as the robot moves? To answer this question, we need to introduce some additional terminology. Let e be a visibility edge whose associated visibility ray intersects the robot at point x. The endpoint of e lying closer to x (possibly x itself) is defined as the *anchor* vertex of e, while the further endpoint is called the *attachment* vertex of e. If a vertex of the shadow (considering the shadow as a polygon) lies in free space, *i.e.*, if it lies inside the bounding polygon and is not on the boundary of an obstacle, then we call it a *free* vertex of the shadow. See Figure 3.20 for an illustration of these definitions.

We now show that as the robot translates, free shadow vertices trace out point conics[1] if their generating edges are anchored on the robot.

Lemma 3.3 *Let u and v be two visibility rays, anchored at robot vertices and tangent to obstacles, such that they intersect to form a free vertex of a shadow. The path traced by the intersection point of rays u and v describes a point conic as the robot translates along a line.*

Proof: Consider the robot at a fixed orientation, with its reference point translating along an edge e of the goal. Let visibility rays u and v be anchored at robot vertices B and C with obstacle attachment points C' and B', respectively. The attachment points B' and C' form fixed points through which the visibility rays u and v must pass. As the robot translates along edge e, line segment BC translates rigidly, with B and C constrained to lie on lines parallel to edge e. What we have constructed corresponds precisely with Maclaurin's construction of a conic [Ped63]:

Lemma 3.4 (Maclaurin 1721) *Let ABC be a variable triangle such that the vertices B and C move on two fixed lines l and m, respectively, and each side BC, CA and AB of the triangle passes through a fixed point. Then the vertex A describes a conic section.*

See Figure 3.21 for an illustration of Maclaurin's construction, and Figure 3.22 for an illustration of the correspondence to our case.

[1] A *point conic* is simply a *conic* (*i.e.*, an ellipse, a parabola, or a hyperbola) that is formed by a locus of intersection points. We use the term "point conic" to distinguish from the dual case of a "line conic", which is formed by the envelope of a set of lines. Formally, a point conic is the locus of the points that are common to pairs of corresponding lines of two pencils between which there is a projectivity.

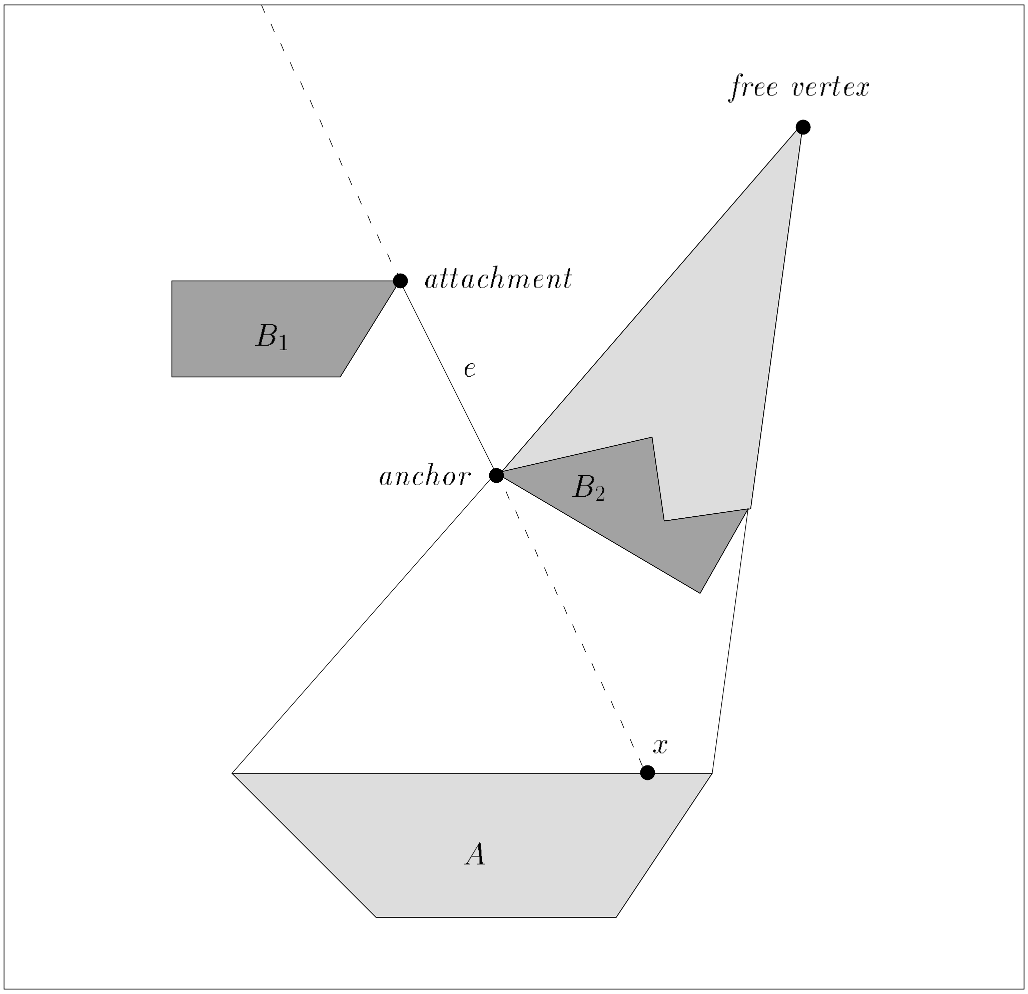

Figure 3.20: The visibility ray for visibility edge e intersects the robot at x. The anchor and attachment vertices are shown. The lightly shaded region represents a shadow; the labeled vertex is a free shadow vertex.

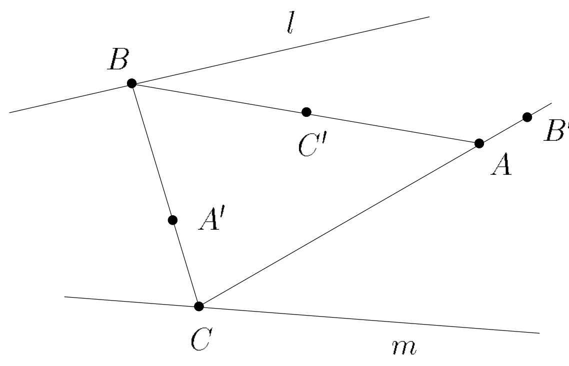

Figure 3.21: Illustration of Maclaurin's construction of a conic. Points A, B, and C define a variable triangle whose edges are constrained to lie on lines passing through fixed points A', B', and C'.

Since parallel lines intersect at infinity in projective geometry, we can put fixed point A' at infinity on the line through B and C, so that the robot can translate. If the three fixed points are colinear, the conic is singular, describing a line through the intersection point of lines l and m. In the case of translational robot motion, three colinear fixed points arise when the line through attachment points C' and B' is parallel to the line through anchor points B and C. The free shadow vertex formed by the intersection of u and v would then describe a line parallel to edge e as the robot translates along e. $\square$

3.4.4 Swept shadows in the weak visibility model

We have shown how to compute shadows for any fixed robot position, and have seen how these shadows change as the robot translates. In order to detect the robot as it enters the goal, we must compute the shadows swept for all positions of the robot in the goal. We define a *swept shadow* of the goal in the weak visibility model to be a maximal connected region of C_{s_p} such that for each point $\mathbf{p}$ in the region, there exists a configuration of the robot in the goal from which the robot is totally occluded.

We compute swept shadows for the robot at a fixed orientation anywhere in the goal by translating the robot polygon along the edges of the goal polygon. The boundary of a swept shadow is composed of obstacle segments and the curves (lines and conics) traced by free vertices. Discontinuities in the boundary of a swept shadow occur at *critical events*. We characterize the critical events as follows:

1. A moving visibility ray becomes aligned with a fixed edge of the visibility graph (see Figure 3.23).

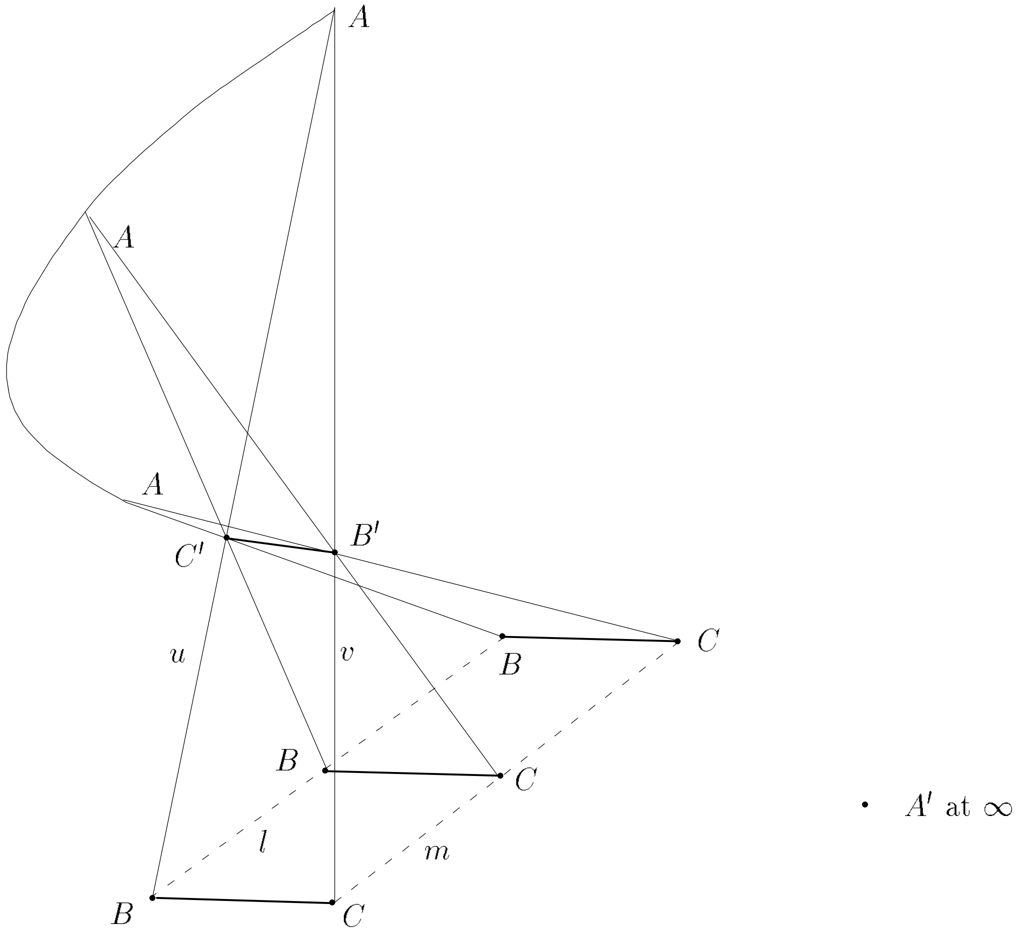

Figure 3.22: Correspondence between Maclaurin's construction and shadows. Segment BC represents an edge of the robot; segment $B'C'$ represents an obstacle edge. The free shadow vertex A describes a point conic, with its supporting edges bounding the shadow. When robot motion is translational, lines l and m are parallel, and fixed point A' lies at infinity.

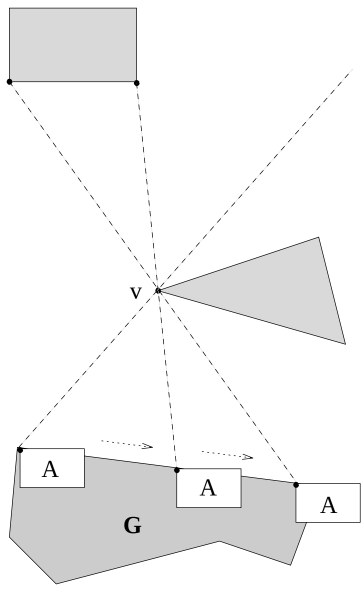

Figure 3.23: The first type of critical event. As A translates in the goal, the visibility ray anchored at v becomes aligned with edges of the visibility graph.

2. A free vertex of a shadow intersects an obstacle edge or the bounding polygon (see Figure 3.24).

3. Two moving visibility rays bounding a shadow become parallel (see Figure 3.25).

Below we present our algorithm for computing the weak visibility region of a robot as it translates through the goal at a known orientation θ. This gives us the set of all sensor placements from which at least one point on the boundary of the robot can be seen, no matter where the robot is in the goal.

Weak visibility algorithm for a translating robot

1. Let e be any edge of goal G. Consider A_θ to be placed on one of the endpoints of e. Call this configuration $\mathbf{q}$. Construct the weak visibility region of robot A at configuration $\mathbf{q}$.

2. Translate A_θ along e. As the shadows cast by the obstacles change, call the area swept out by a shadow a *swept shadow*. Between critical events, the vertices of each shadow move along lines or conics. The equations of these curves can be computed algebraically given the positions of the obstacles in the environment and the visibility rays. Update the boundary of the swept shadows at critical events.

3. Translate A_θ along all other edges e_i, $1 \leq i \leq k$, of G, repeating step 2 for each edge.

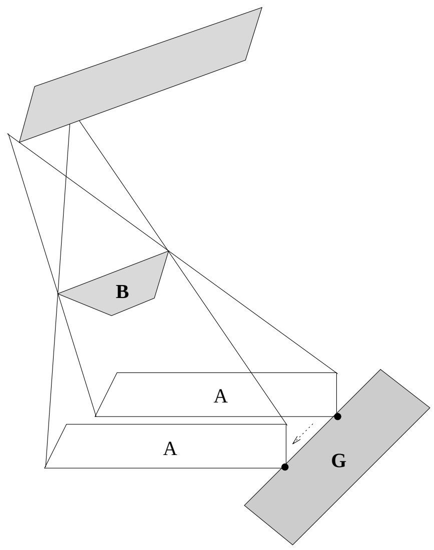

Figure 3.24: The second type of critical event. As A translates in the goal, the shadow cast by obstacle B intersects another obstacle.

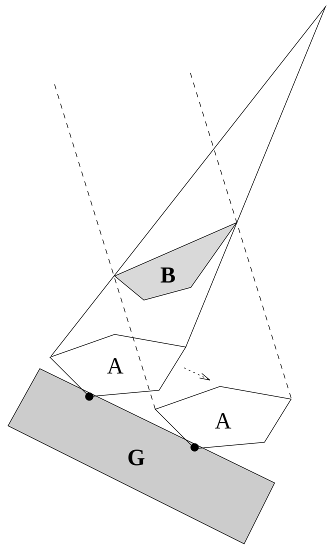

Figure 3.25: The third type of critical event. As A translates in the goal, the edges of the shadow cast by obstacle B become parallel.

4. Compute each swept shadow independently as described in the above steps. The complement of the union of all the swept shadows, the robot, and the obstacles is the weak visibility region.

The output of the algorithm is the set of swept shadows. Note that the boundary of a swept shadow is piecewise linear and conic.

3.4.5 Complexity of the weak visibility algorithm over a region

The extended visibility edges bounding the shadows are all either external local tangents between an obstacle and the robot, or internal local tangents between obstacles. Since the obstacles are fixed, the visibility edges between them remain fixed. As the robot moves, the only visibility edges that move are those that are anchored on a vertex of the robot.

With n vertices in the environment and m robot vertices, there are $O(mn)$ moving visibility edges. As the robot translates along an edge of the goal, a visibility edge anchored at robot vertex a_i and attached at obstacle vertex b_j could become aligned with each of the $O(n)$ fixed visibility edges at obstacle vertex b_j. This gives $O(mn^2)$ critical events of the first type as the robot translates along an edge of the goal. There are $O(m^2n^2)$ free vertices tracing out curves, which may intersect each of the $O(n)$ obstacle segments. This gives $O(m^2n^3)$ critical events of the second type. When the third type of critical event occurs, a free vertex disappears. There are $O(m^2n^2)$ of these events.

At a critical event of the first type, a visibility ray appears or disappears, causing a visibility triangle to appear or disappear. The total cost of handling all updates of this type is $O(mn^3(n+m))$. Only local change is caused by events of the second type and third type.

Between critical events, we simply grow the shadows, either along lines or conics. Note that the shadows never shrink: A point $\mathbf{p} \in C_{s_p}$ is in a shadow with respect to a polygonal goal if there exists some robot configuration such that the robot is not at all visible from $\mathbf{p}$. The computation of swept shadows is done by translating the robot polygon along the edges of the goal, updating the boundary at critical events. The total running time of the algorithm for a goal with k vertices is $O(kmn^3(n+m))$.

3.5 Uncertainty in sensor placement and aim

A real sensor cannot be configured exactly. Rather, it will be subject to both errors in placement and errors in aim. These errors depend on the sensor platform (*e.g.*, a mobile robot). Therefore we would like to compute sensor strategies that take uncertainty in sensor configuration into consideration. In this section, we

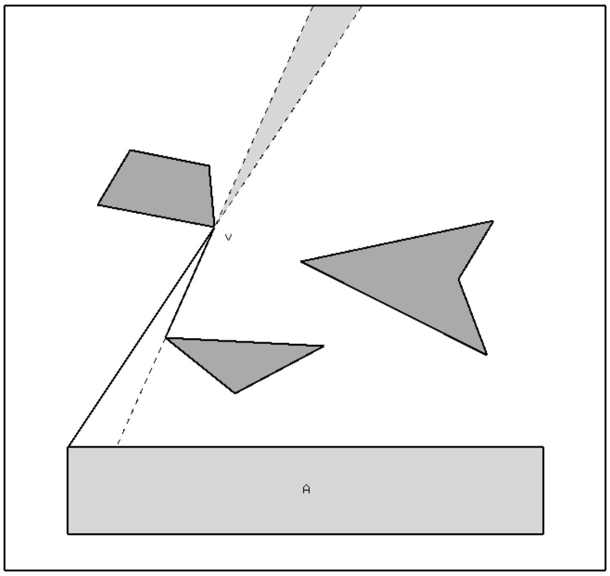

Figure 3.26: A narrow visibility triangle anchored at vertex v is shown lightly shaded.

sketch how the computation of visibility regions can be extended to handle this type of sensor error. Our approach does not yet address the problem of sensor measurement error.

Positional uncertainty characterizes the sensor placement error. Let ϵ_{pos} denote the worst-case positional uncertainty of the sensor. If the commanded sensor placement is $\mathbf{p}$, the actual sensor placement could be any position in the disc of radius ϵ_{pos} centered at $\mathbf{p}$. We handle positional uncertainty by growing the shadows by the uncertainty ball of radius ϵ_{pos}. The complement of the union of these grown shadows and the environment will be the visibility region that accounts for uncertainty in sensor position.

Directional uncertainty characterizes the sensor aim error. Let ϵ denote the maximum angular error of the sensor aim. That is, if the commanded sensing direction is ψ, the actual sensor heading could be any direction in the cone $(\psi - \epsilon, \psi + \epsilon)$. The effect of sensor directional uncertainty is that we must disallow angularly narrow wedges of visibility. This type of uncertainty is most relevant in the case of weak visibility. See Figure 3.26 for an illustration of a narrow visibility triangle. This triangle does not become part of the visibility region when directional uncertainty is considered.

After we compute the visibility rays as described in Section 3.4.1, we visit each vertex in the environment, and combine adjacent visibility triangles that end on the same polygon. We make the following definitions:

1. The maximal union of adjacent visibility triangles anchored on a single vertex v and ending on the same polygon is called a *visibility polygon*. By

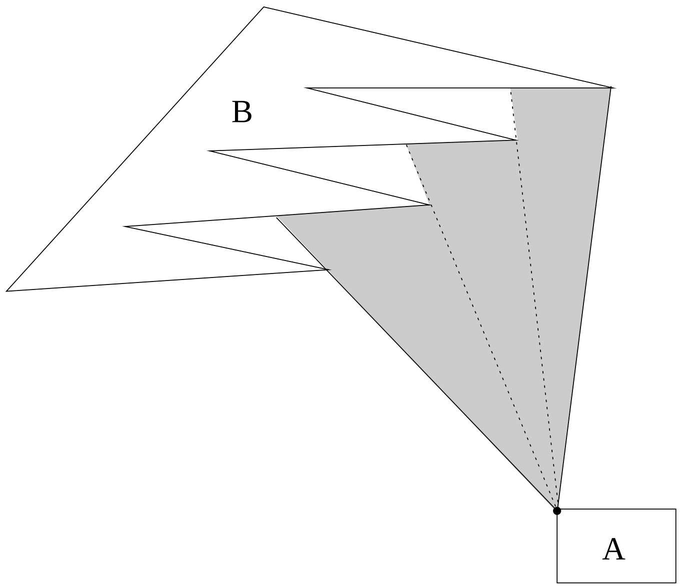

Figure 3.27: Adjacent visibility triangles ending on the same polygon are combined to form a visibility polygon.

construction, visibility polygons are simple.

2. The *core triangle* of a visibility polygon anchored at v is the maximal inscribed triangle whose apex is v.

See Figures 3.27 and 3.28 for illustrations.

If the angle at the apex of such a maximal visibility triangle is less than our angular uncertainty bound ϵ, we discard the polygon. Otherwise, we classify the maximal visibility triangle as an *ϵ-fat triangle*. After this processing, we now have $O(n(n+m))$ fat visibility triangles. We can now use a result of Matoušek *et al.* [MMP+91] on the union of fat triangles. Their result bounds the number of *holes* in a union of fat triangles. In our case, the "holes" are shadows in a union of visibility triangles. Their theorem states that for any fixed $\delta > 0$, and any family $\mathcal{F}$ of n δ-fat triangles, their union has $O(n/\delta^{O(1)})$ holes. When we restrict our visibility triangles to be at least ϵ-fat, we have at most $O((n(n+m))/\epsilon^{O(1)})$ shadows.

When ϵ is a fixed constant, we have at most $O(n(n+m))$ shadows. In effect, this means that considering directional uncertainty actually lowers the complexity of computing the recognizability region.

3.6 Choosing a sensor configuration

So far, our algorithms compute the set of all sensor placements that afford either complete or partial visibility. We call such placements *valid* sensor placements.

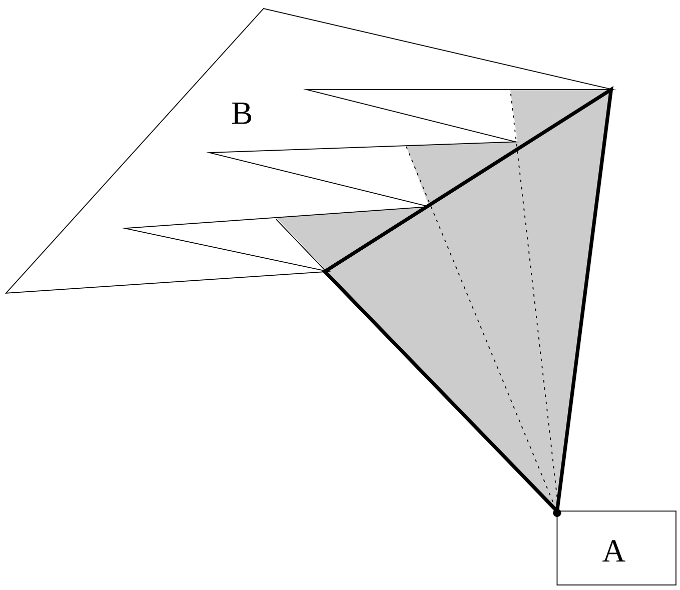

Figure 3.28: The core triangle.

We gave algorithms for a variety of cases, handling different restrictions imposed by the task requirements.

We now briefly address the question of choosing an actual configuration from a computed set of valid sensor placements. Recall that a sensor configuration $\phi \in C_s$ is defined as the pair $(\mathbf{p}, \psi)$ for $\mathbf{p} \in C_{s_p}$ and $\psi \in C_{s_c}$.

3.6.1 Issues in choosing a sensor placement

In choosing a sensor placement within an appropriate visibility region, any of the following types of constraints may need to be considered:

1. Physical constraints such as camera angle and distance constraints of the sensor.

2. Environmental constraints such as requiring that the sensor be mounted on a wall.

3. Task constraints such as the proximity of the new position to the current position (as in active sensing).

For example, if the sensor employed is a CCD camera, the placement must be far enough away from the region to be monitored so that the outermost tangents with the region lie within the viewing cone.

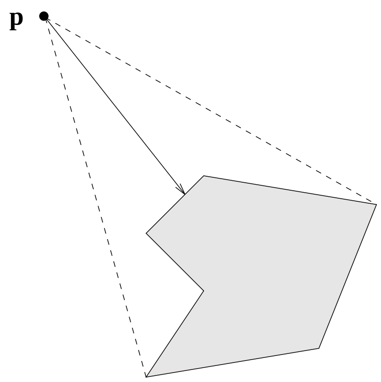

Figure 3.29: Choosing a viewing direction in the complete visibility model. The dashed lines represent tangents from sensor placement **p** to the distinguished polygon (shown shaded). The chosen direction is the solid arrow.

3.6.2 Choosing a sensing direction

Once a sensor placement has been determined, a viewing direction is chosen.

In the case of complete visibility, one of the following methods may be used to determine a viewing direction.

1. If the sensor is a camera, find the outermost tangents between the computed sensor placement and the region to be monitored. Choose the mean angle between these two tangents as the viewing direction. See Figure 3.29 for an illustration.

2. If the sensor is a laser ranger, scan repeatedly across the region to be monitored to detect changes within the region.

For the case of weak visibility, we note that the visibility triangles output by our algorithm implicitly bound the range of viewing directions that afford visibility of the distinguished polygon. To compute a sensor aim given a valid placement **p**, we do the following. First locate a visibility triangle containing **p**. Then use the endpoints of the visibility edges generating that triangle to determine the range of aims. Figure 3.30 illustrates the idea. To compute all viewing directions that afford visibility of the distinguished polygon from sensor placement **p**, we find all visibility triangles containing **p**. Then take the union of the computed ranges of aims over all these triangles.

In our systems work (see Chapter 5) we plan sensing strategies that enable a mobile robot to monitor a particular region in a room. This work relies on the methods presented here for choosing valid sensor placements and aims.

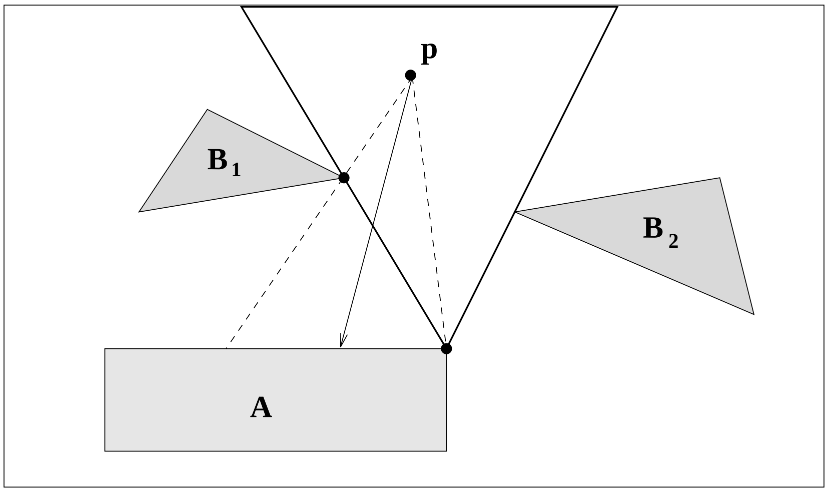

Figure 3.30: Choosing a viewing direction in the weak visibility model. The thick solid lines bound a visibility triangle containing sensor placement **p**. The dashed lines represent the range of viewing directions affording visibility of polygon A from **p**. The black dots on B_1 and A are the endpoints of the visibility edges that generate this range. The solid arrow is a potential chosen direction in this range.

3.7 Discussion

We have presented a framework in which to study sensor configuration, and precise algorithms to compute sensor placements. This work is based on the use of visibility as a model of detectability. We justify this model by noting that, for most sensors, the region from which the robot can be detected is a subset of the region from which it is visible. We have presented algorithms for computing the set of sensor placements enabling detection of a stationary or moving polygon in both a complete and weak model of visibility. The algorithms for the stationary case have both been implemented (see Sections 5.1.1 and 5.1.2). The implementation of the *complete visibility algorithm for a stationary robot* was used as part of a demonstration system of robot surveillance. The system plans and executes sensing strategies that enable a mobile robot equipped with a CCD camera to monitor a particular region in a room.

In the next chapter we investigate an application of the algorithms presented here to the *Error Detection and Recovery Framework* [Don89]. We sketch how the techniques developed here for computing visibility regions can be applied to the problem of distinguishing between success and failure in EDR planning. In Chapter 5 we describe our implementations of the algorithms presented in Sections 3.3.1 and 3.4.1 for computing complete and partial visibility regions for a stationary polygon. The robot surveillance project is described in Section 5.2.

Chapter 4

Sensor Configuration in EDR

In this chapter, we briefly examine an application of our work in sensor configuration planning to the Error Detection and Recovery framework [Don89]. An *Error Detection and Recovery* (EDR) strategy is one that is guaranteed to reach the goal when the goal is recognizably reachable, and signals failure otherwise. Given a start region and a motion strategy θ, an EDR planner finds a failure region H such that H is recognizably reachable, but such that it is impossible to enter G from H [Don89]. See Figure 4.1 for an illustration of the EDR regions for a peg-in-hole task. Figure 4.2 gives an example of the problem we would like to solve. The EDR framework guarantees that under generalized damper dynamics, the robot will eventually reach G or H. Furthermore, having entered G or H, it will never leave. Given this guarantee of reachability, we wish to strengthen it to a guarantee of recognizability: we want to know which of G and H has been attained. We will show how a sensor can be configured so that it can distinguish between the robot in G and the robot in H.

Here we consider the case of translational motion in a planar polygonal environment. Denote robot A at configuration $\mathbf{q} \in C_r = \mathbb{R}^2$ by $A_{\mathbf{q}}$. We wish to find a sensor placement $\mathbf{p} \in C_{s_p} = \mathbb{R}^2$ and a sensor aim $\psi \in C_{s_c} = S^1$. We call our sensor configuration space $C_s = C_{s_p} \times C_{s_c} = \mathbb{R}^2 \times S^1$.

We have a general sensor which, when placed at sensor position $\mathbf{p} \in C_{s_p}$ and pointed in a direction $\psi \in C_{s_c}$, is sensitive to a region which we call the *sensitive volume* $SV(\mathbf{p}, \psi) \subset C_r$. This assumes a very general sensor model; it says that the range of readings returned by a sensor at configuration $(\mathbf{p}, \psi)$ is specified by the set $SV(\mathbf{p}, \psi)$ (see Figure 3.2).

A different approach to the incorporation of sensor planning in the EDR framework was first presented by Donald [Don89]. In that approach, an equivalence is established between sensing and motion in configuration space. *Active sensing* for a mobile robot is reduced to motion, by exploiting the similarity between visibility and generalized damper motions. In contrast, we present a framework that is closer to actual sensors. We rely on the visibility algorithms presented in Chap-

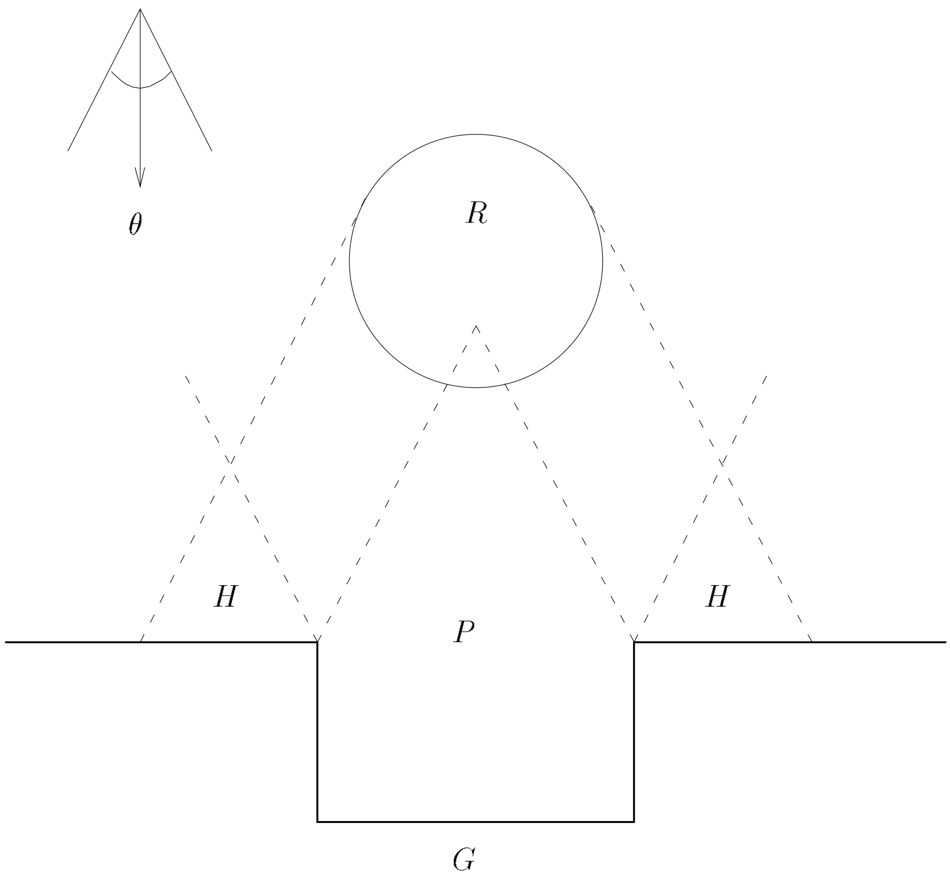

Figure 4.1: EDR regions for the peg-in-hole task in configuration space. θ is the direction of the commanded motion. The actual motion is subject to uncertainty as indicated by the error cone. R is the start region. G is the goal region and H is the failure region. P is the preimage of the goal.

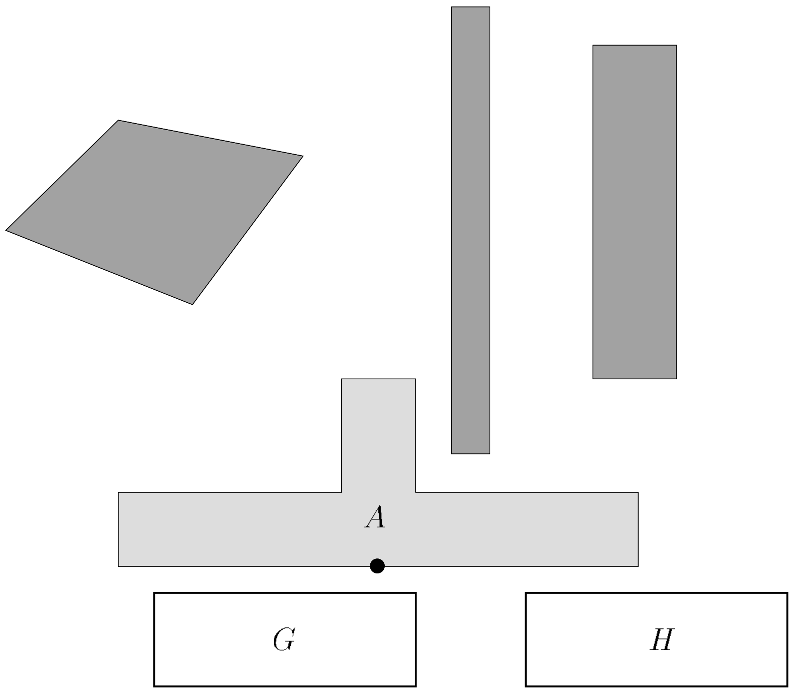

Figure 4.2: An example setup for the problem of sensor configuration in EDR. A represents the robot; its reference point is indicated by the black dot. G is the goal region and H is the failure region. The darkly shaded polygons are obstacles. The problem is to find a sensor placement from which $A \in G$ and $A \in H$ can be distinguished.

ter 3, which have been implemented and used with mobile robots in the lab (see Chapter 5).

4.1 Recognizability and confusability

By definition, an EDR plan achieves either G or H, and must be able to report which of G or H has been reached. We develop a method here of determining how a sensor should be configured so that this goal recognizability can be achieved. The basic idea is that the sensor should be positioned in such a way that attainment of the goal G can be recognized, attainment of the failure region H can be recognized, and attainment of G and H cannot be confused.[1] That is, given that we know robot A is in $G \cup H$, we can determine which of G or H contains A.

For a region $X \subset C_r$, let $R(X)$ denote its *recognizability* region, that is, the set of all sensor placements from which the sensor can return a reading for an object A in region X. Let $C(X, Y)$ denote the *confusability* region, that is, the set of all sensor placements from which the sensor cannot tell $A \in X$ and $A \in Y$ apart.

To generate recognizability and confusability regions as part of an EDR strategy, we compute $R(G)$, $R(H)$, and $C(G, H)$. We wish to find a sensor placement $\mathbf{p} \in C_{s_p}$ such that $\mathbf{p} \in R(G) \cap R(H) - C(G, H)$. See Figure 4.3 for an example setup of this problem. Figure 4.4 illustrates a case in which $A \in G$ and $A \in H$ may be confused.

In the next section, we outline a way of characterizing recognizability and confusability regions in terms of configuration space obstacles for the case of an idealized ranging sensor. We sketch algorithms for computing recognizability and confusability regions for general sensors in Section 4.2.

4.1.1 A configuration space approach

Consider the following model of an idealized laser ranger. When at sensor configuration $(\mathbf{p}, \psi)$, the sensor returns a distance reading d and a surface normal reading α. See Figure 4.5 for an illustration. Both readings are accurate to within some known bound. Given this sensor model, recognizability and confusability regions can be expressed in the language of configuration space obstacles. To review some definitions and conventions [LW79,Loz83]:

- The robot is denoted by A, and the robot at configuration $\mathbf{q}$ by $A_{\mathbf{q}}$.
- Obstacle i is denoted by B_i, and robot configuration space by C_r.
- $CO^r_A(B)$ is the set of robot configurations in which A overlaps B.

[1] This is similar to the notion introduced by Buckley [Buc87] of a *confusable set* in the context of motion planning. There, two contact points x and y are said to be *confusable* if they are capable of generating the same position and force measurements.

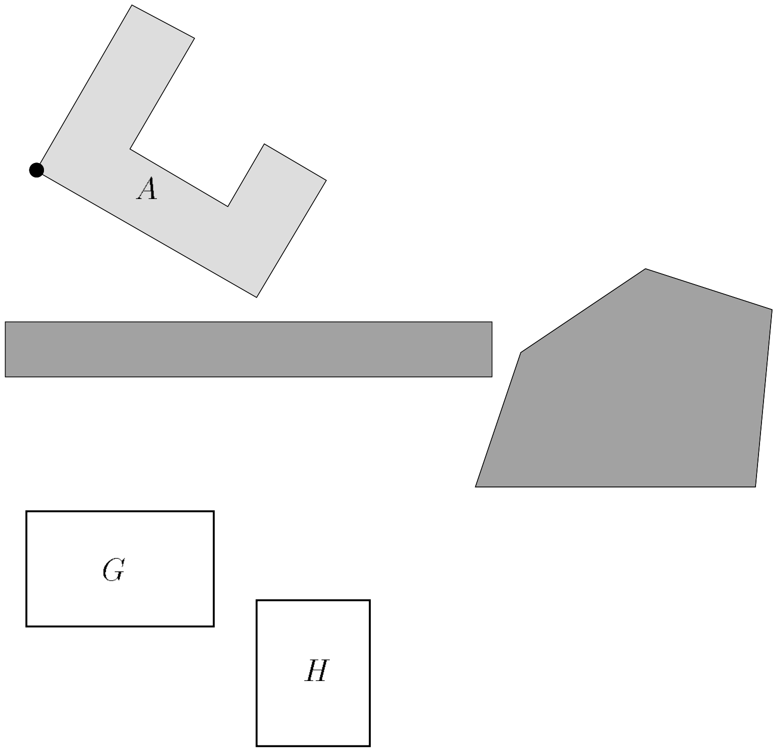

Figure 4.3: An example setup for the problem of confusability. A represents the robot; its reference point is indicated by the black dot. G is the goal region and H is the failure region. The darkly shaded polygons are obstacles.

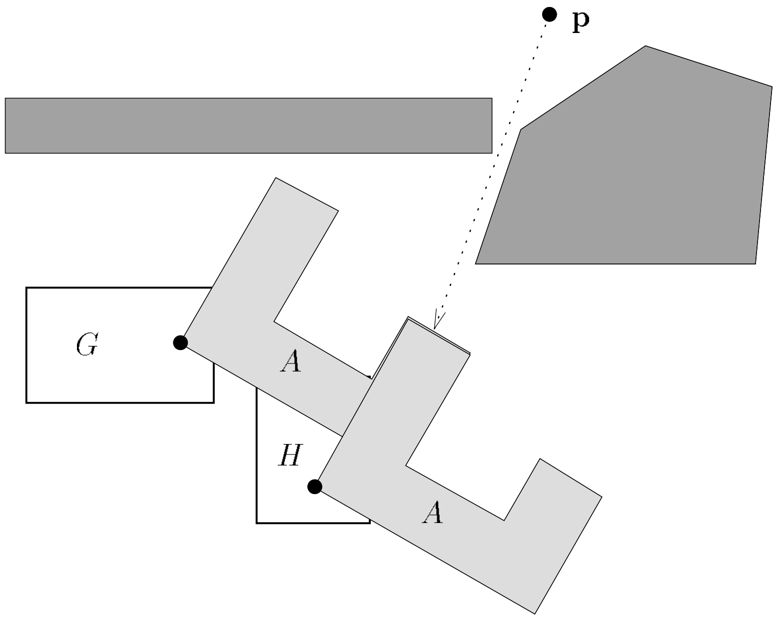

Figure 4.4: $A \in G$ and $A \in H$ are confusable from sensor placement $\mathbf{p}$.

- $CI^r_A(B)$ is the set of robot configurations in which A lies completely inside B.

Note that $C_r - CO^r_A(B) = CI^r_A(C_r - B)$.

Given sensor configuration $\phi = (\mathbf{p}, \psi) \in C_s$, let $SV(\phi)$ denote the region to which the sensor at ϕ is sensitive. See Figure 4.6 for an illustration. For robot configuration $\mathbf{q} \in C_r$, let $L(\mathbf{q}, \phi)$ denote the line segment from the sensor at configuration ϕ to the first contact with $A_{\mathbf{q}}$. See Figure 4.7. Using these definitions, we can now represent recognizability and confusability regions in terms of c-space obstacles. Following the above definitions we can define $CO^s_{SV(\phi)}(A_{\mathbf{q}})$ to be the set of sensor configurations ϕ such that $SV(\phi)$ intersects $A_{\mathbf{q}}$, and $CO^s_{L(\mathbf{q},\phi)}(B_i)$ to be the set of sensor configurations ϕ such that $L(\mathbf{q}, \phi)$ intersects B_i. For a given $\mathbf{q}$, it follows that

$$CO^s_{SV(\phi)}(A_{\mathbf{q}}) \cap (\bigcap_i -CO^s_{L(\mathbf{q},\phi)}(B_i))$$

is the set of sensor configurations from which $A_{\mathbf{q}}$ can be detected with a laser ranger. Thus the set $R(X)$ of configurations that enable detection of A over all $\mathbf{q}$ in region $X \subset C_r$ is

$$R(X) = \bigcap_{\mathbf{q} \in X} [CO^s_{SV(\phi)}(A_{\mathbf{q}}) \cap (\bigcap_i CI^s_{L(\mathbf{q},\phi)}(-B_i))].$$

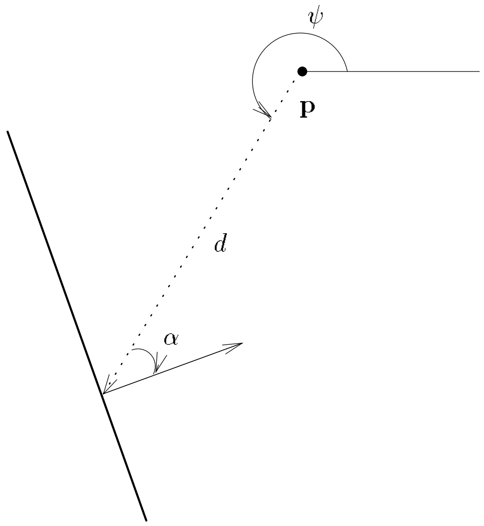

Figure 4.5: Illustration of the idealized laser ranger model. The sensor is placed at position $\mathbf{p}$ and pointed in direction ψ. It returns a distance reading d and a surface normal reading α.

Since there may exist some intersection between $R(G)$ and $R(H)$, we must compute the set of sensor configurations which may lead to *confusion* of G and H. Define $\partial_\phi A_\mathbf{q}$ to be the boundary of $A_\mathbf{q}$ visible while the sensor is in configuration ϕ. Let $\ell(\mathbf{q}, \phi)$ denote the endpoint of $L(\mathbf{q}, \phi)$ such that $\bigcup_{\mathbf{q}\in G} \ell(\mathbf{q}, \phi) = \bigcup_{\mathbf{q}\in G} \partial_\phi A_\mathbf{q}$. Then the set $C(G, H)$ of sensor configurations that may lead to confusion of $A \in G$ and $A \in H$ is

$$C(G,H) = \{\phi \mid \bigcup_{\mathbf{q}\in G} \ell(\mathbf{q},\phi) \cap \bigcup_{\mathbf{q}\in H} \ell(\mathbf{q},\phi) \neq \emptyset\}.$$

Thus given that the robot is in a configuration in $G \cup H$, we can place a laser range sensor in a configuration $\phi \in R(G) \cap R(H) - C(G, H)$ such that goal attainment or failure can be recognized.

4.2 Computing recognizability and confusability regions

In this section we sketch how to compute recognizability and confusability regions for a general sensor. For robot A at configuration $\mathbf{q}$, we can express the set of all sensor configurations from which $A_\mathbf{q}$ is recognizable as:

$$R_\mathbf{q} = \{(\mathbf{p}, \psi) \mid A_\mathbf{q} \cap SV(\mathbf{p}, \psi) \neq \emptyset\}.$$

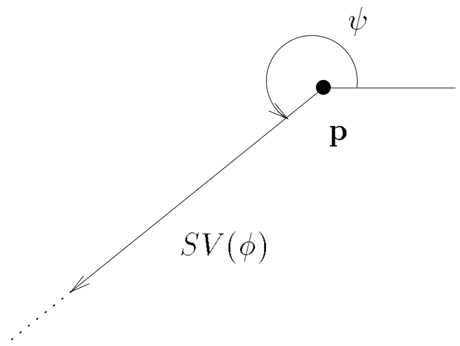

Figure 4.6: The sensitive volume for a laser ranger. The sensor is placed at position $\mathbf{p}$ and pointed in direction ψ. The sensitive volume is the ray $SV(\phi)$.

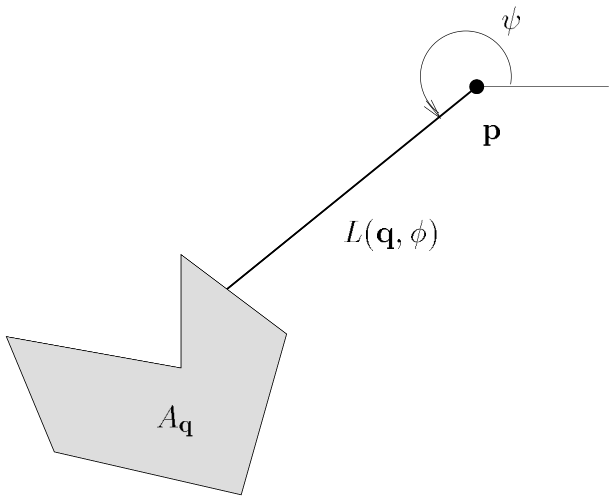

Figure 4.7: $L(\mathbf{q}, \phi)$ is the line segment from the sensor at configuration $\phi = (\mathbf{p}, \psi)$ to the first contact with $A_{\mathbf{q}}$.

The set of all sensor *placements* from which $A_\mathbf{q}$ is recognizable is:

$$\hat{R}_\mathbf{q} = \{\mathbf{p} \mid (\exists\psi) : (\mathbf{p}, \psi) \in R_\mathbf{q}\},$$

which is the projection of $R_\mathbf{q}$ from $C_s = \mathbb{R}^2 \times S^1$ onto $C_{s_p} = \mathbb{R}^2$. $\hat{R}_\mathbf{q}$ is the region in C_{s_p} from which robot A, at configuration $\mathbf{q}$, is partially visible. We use the *weak visibility algorithm for a stationary robot* presented in Section 3.4.1 to compute it.

Note that $R_\mathbf{q}$, the set of all sensor *configurations* from which $A_\mathbf{q}$ is recognizable, is implicitly represented by the output of the weak visibility algorithm: for a given placement $\mathbf{p} \in \hat{R}_\mathbf{q}$, we can compute the set of sensor aims $\{\psi \mid (\mathbf{p}, \psi) \in R_\mathbf{q}\}$ given the visibility triangles containing $\mathbf{p}$ (see Section 3.6).

Extending $\hat{R}_\mathbf{q}$ to a region $X \subset C_r$, we define:

$$\hat{R}(X) = \{\mathbf{p} \mid (\forall \mathbf{q} \in X)(\exists\psi) : A_\mathbf{q} \cap SV(\mathbf{p}, \psi) \neq \emptyset\}.$$

To test for goal recognizability we substitute G for X in the above definition. $\hat{R}(G)$ is the region in C_{s_p} from which robot A, when in G, will always be at least partially visible. We use the *weak visibility algorithm for a translating robot* from Section 3.4.3 to compute the sets $\hat{R}(G)$ and $\hat{R}(H)$.

We now define the set of all sensor configurations that could lead to confusion of G and H. The set $C(G, H)$ is the set of sensor configurations from which the same sensor readings are returned for a robot configuration in G and H:

$$C(G, H) = \{\phi \mid (\exists \mathbf{q} \in G, \mathbf{q}' \in H) : (SV(\phi) \cap A_\mathbf{q}) = (SV(\phi) \cap A_{\mathbf{q}'})\}.$$

We define $\hat{C}(G, H)$ to be the set of sensor *placements* from which all viewing directions lead to confusion of G and H:

$$\hat{C}(G, H) = \{\mathbf{p} \mid (\forall\psi) : (\mathbf{p}, \psi) \in C(G, H)\}.$$

Note that a sensor reading that confuses a robot $A_\mathbf{q}$ in G with a robot $A_{\mathbf{q}'}$ in H is due to an edge of $A_\mathbf{q}$ being colinear with an edge of $A_{\mathbf{q}'}$. See Figure 4.8 for an example (see also Figure 4.4).

Here is a sketch of how $\hat{C}(G, H)$ can be computed:

1. For each edge orientation α of A consider the set of all edges of A with that orientation. Call this set E_α. Compute all possible placements of these edges by convolving E_α with G and H:

$$E_\alpha \oplus G = \bigcup_{e \in E_\alpha} e \oplus G$$

and

$$E_\alpha \oplus H = \bigcup_{e \in E_\alpha} e \oplus H.$$

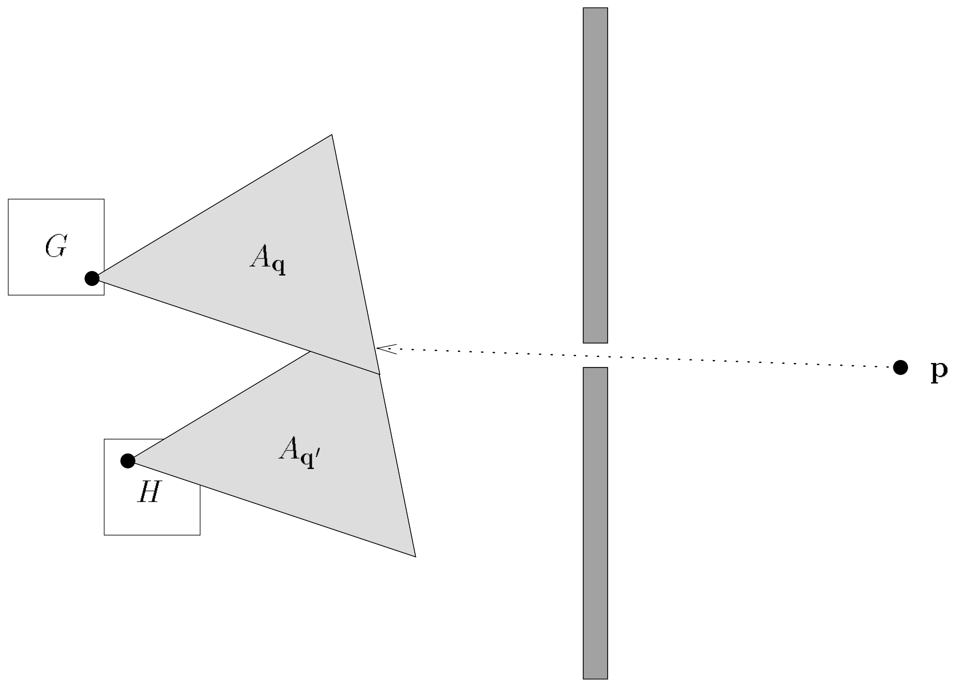

Figure 4.8: A sensor reading that confuses $A_{\mathbf{q}} \in G$ and $A_{\mathbf{q}'} \in H$ is due to an edge of $A_{\mathbf{q}}$ being colinear with an edge of $A_{\mathbf{q}'}$. The darkly shaded rectangles are obstacles.

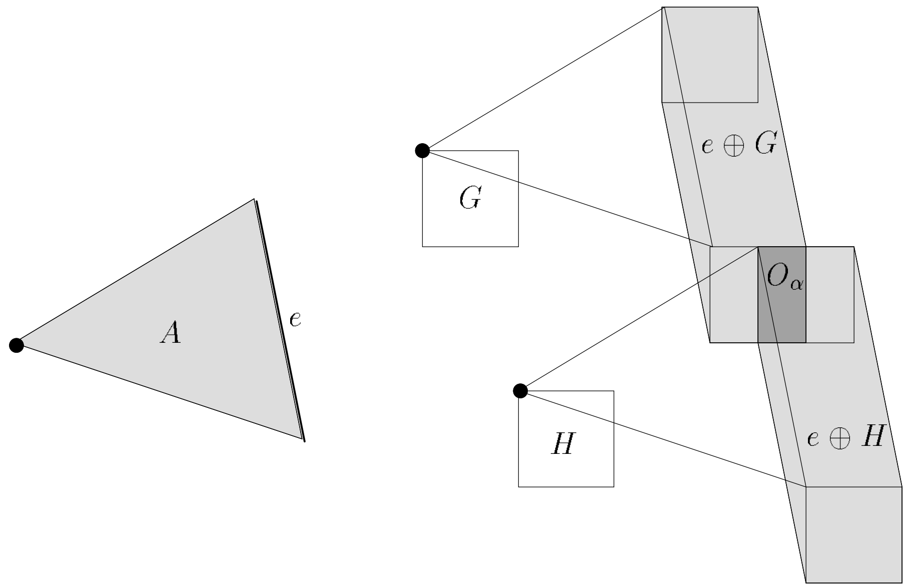

Figure 4.9: Edge e at orientation α of robot A is convolved with G and H. The darkly shaded region is the overlap O_α. Sensor readings in O_α can lead to confusion of G and H.

2. Compute the *overlap* $O_\alpha = (E_\alpha \oplus G) \cap (E_\alpha \oplus H)$ of these regions. See Figure 4.9 for an example using the setup of Figure 4.8.

3. Compute the set of placements from which only O_α is visible using the techniques introduced in Chapter 3. Call this set C_α. C_α is the set of all sensor placements from which edges in E_α can be confused. See Figure 4.10 for an illustration.

4. Finally, $\hat{C}(G, H)$ is the union of the C_α for all edge angles α of A, *i.e.*,

$$\hat{C}(G, H) = \bigcup_\alpha C_\alpha.$$

Now, $\hat{R}(G) \cap \hat{R}(H) - \hat{C}(G, H)$ is the set of all sensor placements from which attainment of the goal G can be recognized, attainment of the failure region H can be recognized, and attainment of G and H cannot be confused.

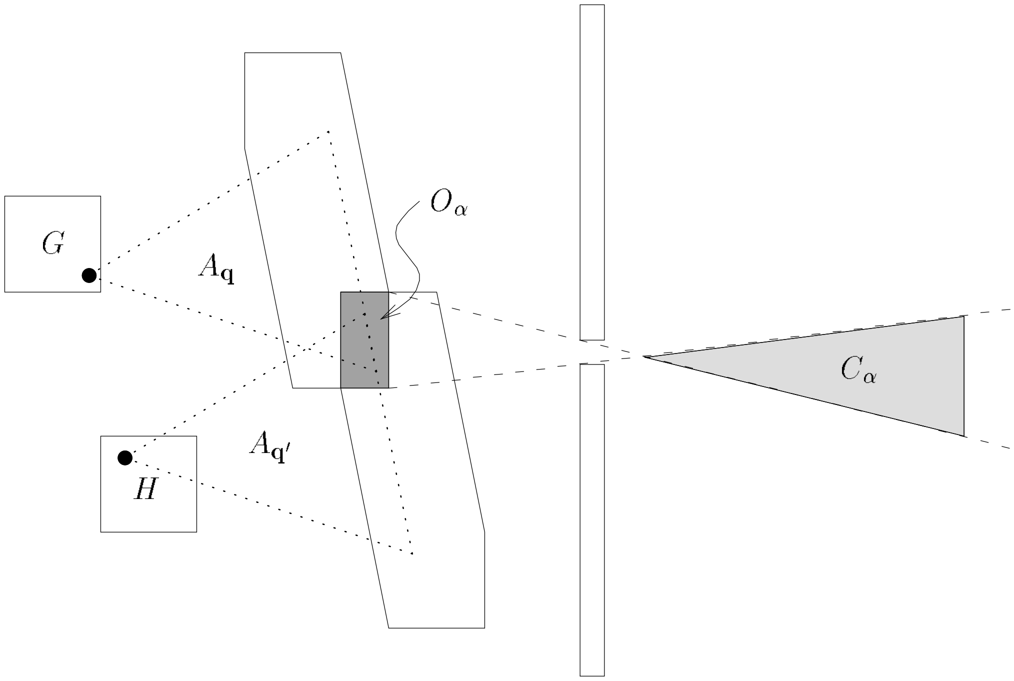

Figure 4.10: The darkly shaded region is the overlap $O_\alpha = (e \oplus G) \cap (e \oplus H)$. The lightly shaded region is C_α, the set of sensor placements from which only O_α is visible.

4.3 Discussion

In this chapter, we have sketched an application of the visibility algorithms described in Chapter 3 to the problem of distinguishing attainment of the goal or failure region as part of an error detection and recovery strategy. Given a general sensor model, we defined *recognizability* and *confusability* sets in terms of the *sensitive volume* of the sensor. We then sketched ways to efficiently compute representations of these sets, using the techniques already developed in Chapter 3. In the next chapter we describe our implementations of the visibility algorithms of Chapter 3, and how we have used these algorithms with mobile robots in the lab. We expect that straightforward extensions to these systems could be made to plan sensing strategies within the EDR framework.

Chapter 5

Implementations and Experiments

In this chapter we discuss our implementations and experimental results. The algorithms for computing the complete visibility region of a polygon and the partial (or weak) visibility region of a polygon have both been implemented. These implementations are discussed in Section 5.1. The complete visibility algorithm was used to conduct experiments using two of the mobile robots in the Cornell Robotics and Vision Laboratory. This work led to the development of a robot surveillance demonstration project, which we discuss in Section 5.2.

5.1 Implementation of the visibility algorithms

We now discuss our implementations of the algorithms given in Section 3.3 for computing the complete visibility region and in Section 3.4 for computing the partial visibility region of a stationary polygon. The implementations closely follow the algorithmic descriptions given in Chapter 3. However, they do not realize the asymptotic complexity bounds reported there, since we use simpler algorithms for some parts. Both algorithms are implemented in Scheme [CR92], and are built on top of existing packages for graphics, geometric modeling, and plane sweep.

The graphics package we use, called MTV, was developed by Patrick Xavier and Michael Erdmann. This system provides an interface to the XWindows drawing routines in the XLIB library. It includes basic routines for initializing a graphics window, drawing objects, capturing mouse input, *etc.* The system is implemented in Common Lisp and requires the XLIB package.

The geometric modeling system we used was developed by Jonathan Rees. This system provides high-level constructs for dealing with geometric entities, called *cells*. Cells are either vertices, edges, or faces. The system is actually a $3D$ geometric modeling package, based on the EIGHT system for geometric modeling and three-dimensional graphics developed by Bruce Donald [Don91]. We use it to model planar geometric objects by defining a two-dimensional view so that objects are displayed as if being viewed from above. Vertices are represented as three-space vectors in homogeneous coordinates. In our case, all vertices have a z-coordinate of 0. The system is written in Scheme and requires graphics capabilities such as those provided by MTV. The particular Scheme implementation we use is Pseudoscheme [CR92] running in Lucid Common Lisp with XLIB. The graphics module (win.lisp) provides a front end to some of the routines in the MTV and XLIB packages.

The plane sweep module, called LIMITED, was written by Bruce Donald and John Canny [Don89]. This system provides software for computing arrangements, backprojections and forward projections, using as its computational engine a plane sweep algorithm for computing arrangements of polygons. We use the plane sweep code for computing an arrangement in the complete visibility implementation.

The implementations of the complete visibility and partial visibility algorithms are fundamentally different, but rely on some common routines. Input to both systems is a *scene*, consisting of a list of obstacles, a distinguished polygon, and a bounding polygon. Visibility regions are computed for the distinguished polygon in the presence of the obstacles. The distinguished polygon can be considered to be either a robot, a goal, or a polygonal approximation to the convolution of a robot with a goal. That is, both systems compute visibility regions for a stationary polygon. If we were to extend the systems to enable the computation of visibility regions for a moving robot, we would consider the distinguished polygon to represent the robot. For ease of exposition, we refer to this polygon as the "goal" in the following discussion.

5.1.1 Implementation of the partial visibility algorithm

The *weak visibility algorithm for a stationary polygon* described in Section 3.4 is implemented in a straightforward manner. First the visibility graph of the entire scene is computed. Our implemented algorithm for computing the visibility graph allows for both convex and non-convex polygons. The naive method we use here can be quite slow on an environment with many vertices, making the visibility graph computation the slowest part of the system. This could be improved for efficiency by implementing a more sophisticated visibility graph algorithm such as the optimal algorithms presented in [GM87,AAG$^+$86,Wel85].

Once the visibility graph has been computed, its edges are extended in both directions. All extended edges that do not intersect the goal are discarded. The extended edges that intersect the goal are then clipped to the segment with one endpoint on the goal and the other on an edge of the scene. We call the resulting edges *visibility rays*. That is, each visibility ray is the maximal-length segment of an extended edge inside the bounding polygon not intersecting the interior of an obstacle. The collection of all visibility rays is called the *extended visibility graph*. See Figure 5.1 for an illustration of a simple visibility graph and Figure 5.2 for its

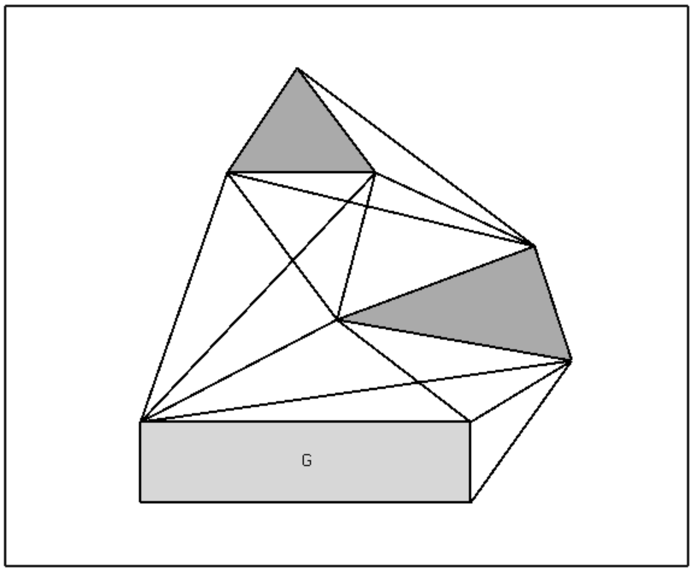

Figure 5.1: A simple visibility graph.

extended visibility graph.

Each visibility ray is maintained as a list of the following components:

1. an endpoint on the goal (referred to as the *goal point*)
2. an endpoint in the environment (referred to as the *end point*)
3. the angle of the visibility ray with respect to the x-axis
4. the endpoints of the original visibility edge along with the faces to which they are adjacent (referred to as the *close point* and *far point,* respectively.)

Note that the *goal* point and *close* point may coincide, as may the *far* point and *end* point. See Figures 5.3 and 5.4 for illustrations.

Below is the top-level routine for computing the extended visibility graph.

We employ the following stylistic conventions:

- "`A->B`" indicates a type conversion between objects `A` and `B`.
- names of boolean predicates end with "`?`".
- names of magic constants begin and end with "`*`".

Recall that a *scene* consists of the following:

1. the obstacles of the environment, returned by the expression `(obstacles scene)`.
2. the goal of the environment, returned by the expression `(goal scene)`.
3. the bounding polygon of the environment, returned by the expression `(bbox scene)`. Note that the bounding polygon need not be a rectangle.

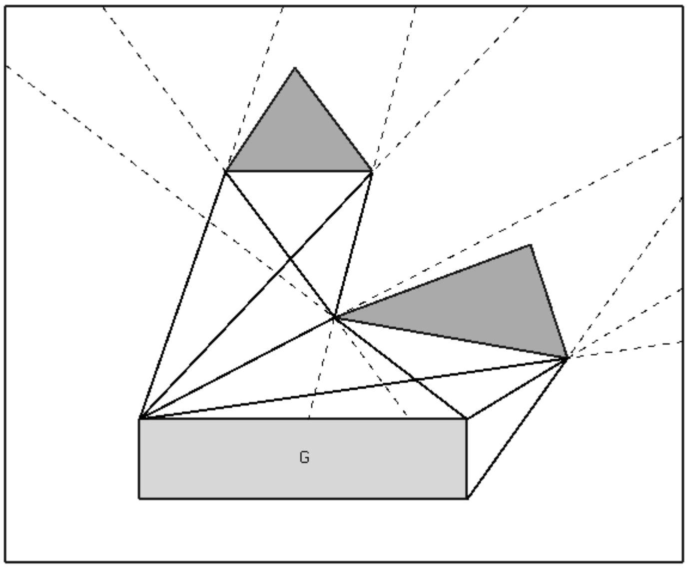

Figure 5.2: The extended visibility graph.

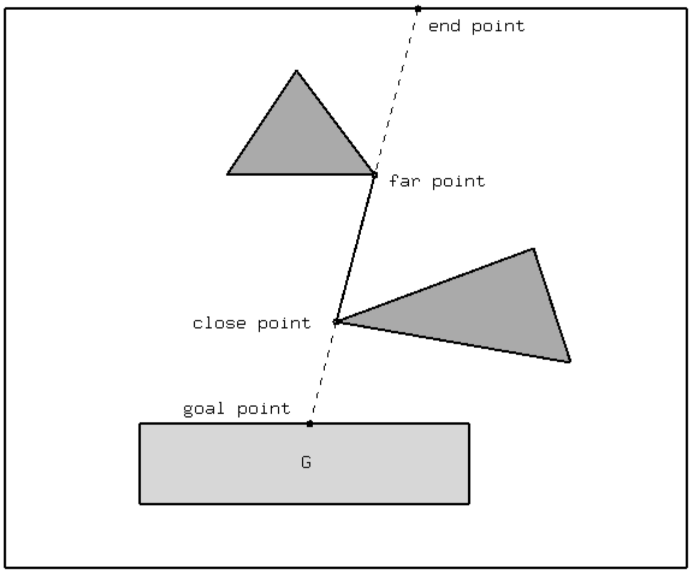

Figure 5.3: A visibility ray.

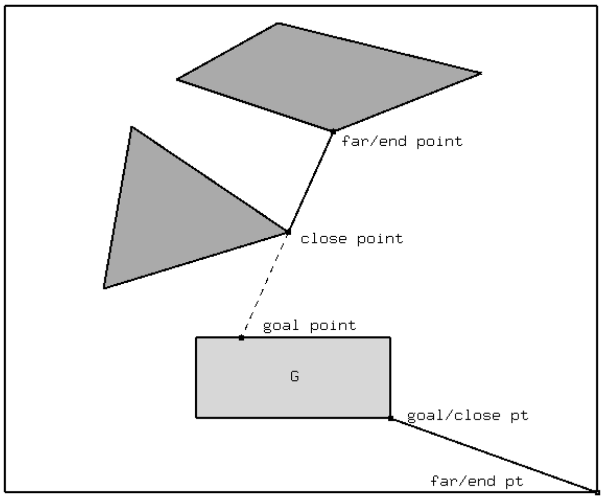

Figure 5.4: Two visibility rays with coinciding labels.

```
(define (extended-vg vg scene)
  (cond ((null? vg) '())
        (#t
         (let* ((vis-edge (car vg))
                ; orient the visibility edge away from the goal
                (edge (orient-from-poly (vis-edge->edge vis-edge)
                                        (goal scene)))
                ; extend into a ray starting at the goal
                (ray (ray-from-poly edge (goal scene)))
                (front (make-edge (edge-head edge) (edge-head ray)))
                (back (make-edge (edge-tail ray) (edge-head edge))))
           ; test to see if the ray is a valid one
           (if (and (poly-intersect? ray (goal scene) *endpts-OK*)
                    (not (edge-of-poly? edge (goal scene)))
                    (not (poly-intersect? edge (goal scene) *no-endpts*))
                    (in-free-space? back (cons (bbox scene)
                                               (obstacles scene))))
               ; compute the components of the visibility ray
               (let* ((goal-pt (edge-tail ray))
                      (end-pt (find-endpt
                                  front (obstacles scene) (bbox scene)))
                      (angle (edge-angle edge))
                      (vis-edge2 (cons edge (vis-edge->faces vis-edge)))
                      (vis-ray (list goal-pt vis-edge2 end-pt angle)))
                 (cons vis-ray (extended-vg (cdr vg) scene)))
             (extended-vg (cdr vg) scene))))))
```

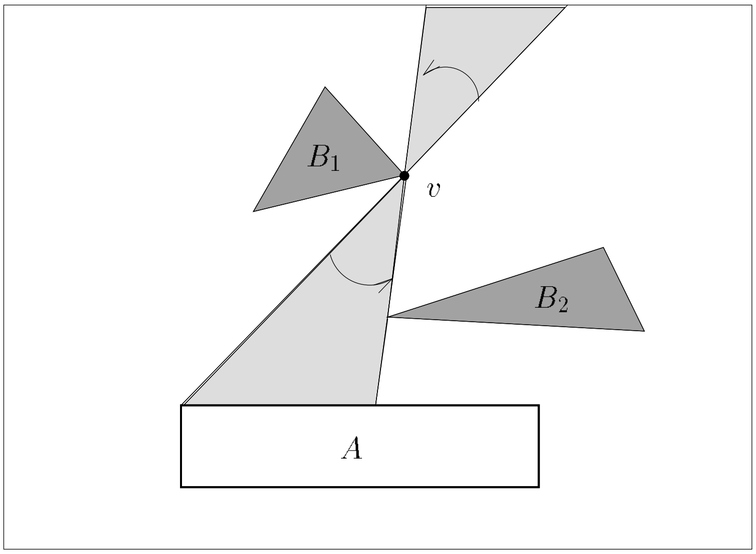

Figure 5.5: An angular sweep between two visibility rays at vertex v. The lightly shaded regions are visibility triangles.

Given the representation described above of the extended visibility graph, we compute visibility triangles. Each vertex of the scene is visited, and the visibility rays at that vertex are sorted by their angle. We then perform an angular sweep of the visibility rays. Figure 5.5 gives an illustration. Sweeping from one visibility ray to the next in the sorted list forms two triangles having the current vertex (vertex 1 in Figures 5.6–5.13) at their apex. One triangle has its other endpoints on the goal, and one has its other endpoints on an edge of the scene further from the goal. We call these triangles lower and upper triangles, respectively. Endpoints 2 and 3 of the upper triangle are found by a case analysis on the points of adjacency and endpoints of the two consecutive extended visibility rays under consideration. Determining the second point depends only on the current ray and whether it is blocked from rotating counterclockwise. Figures 5.6–5.9 show some possible cases for determining the second point of the triangle. In the first case, the visibility ray is blocked from sweeping counterclockwise by the face adjacent to the *far* point. So here the *far* point is chosen as the second point. In all other cases, the *end* point is chosen.

Figures 5.10–5.13 illustrate some of the cases for determining the third point of the triangle, when the anchor vertex is the *close* point of the first ray. The cases for when the anchor vertex is the *far* point are similar. The following rule for determining the third point holds for all cases: The *far* point of the next visibility ray in the sorted list will be the third point if it lies along the same obstacle edge as the second point, as in Figures 5.10–5.11. Otherwise the *end* point of the next ray will be the third point, as in Figures 5.12–5.13.

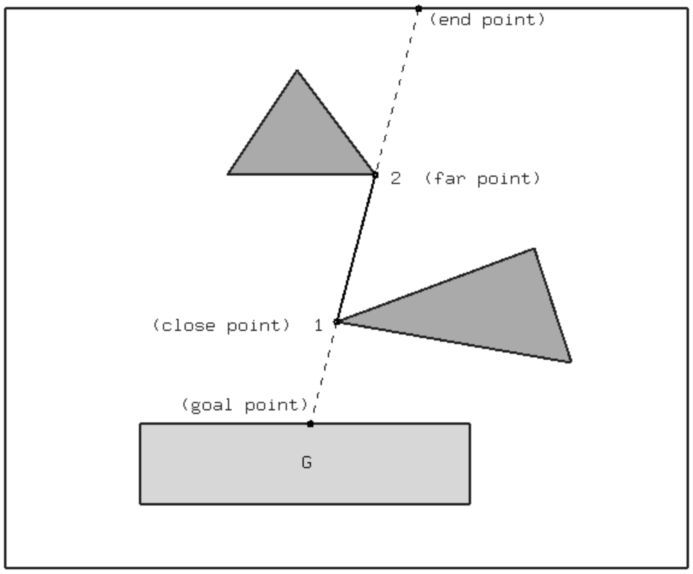

Figure 5.6: Determining the second point of the triangle: a case with the apex at the *close* point. The ray is blocked from sweeping.

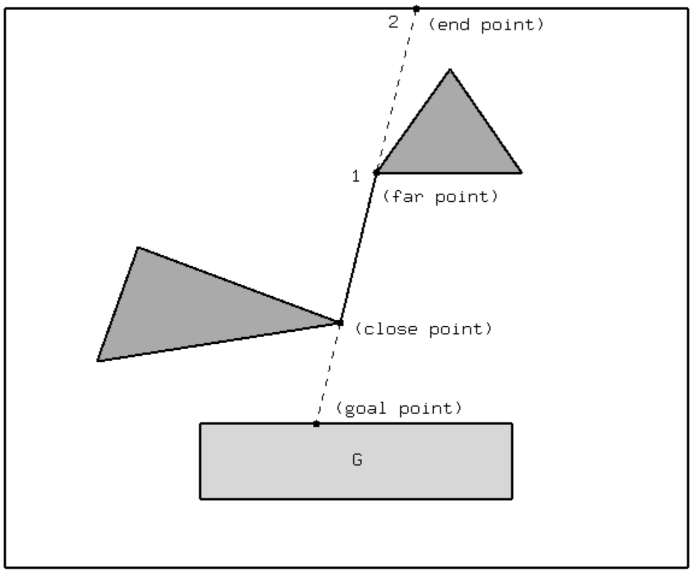

Figure 5.7: Determining the second point of the triangle: a case with the apex at the *far* point and an obstacle to the right. The ray can sweep counterclockwise around vertex 1.

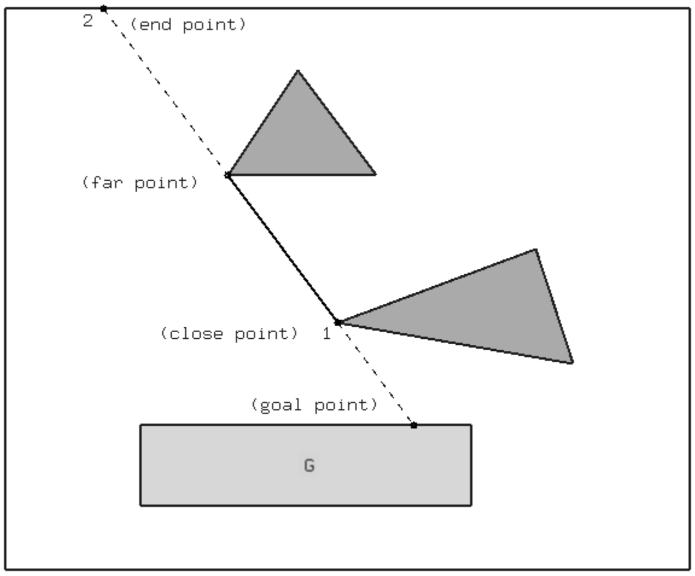

Figure 5.8: Determining the second point of the triangle: a case with the apex at the *close* point. The ray can sweep counterclockwise around vertex 1.

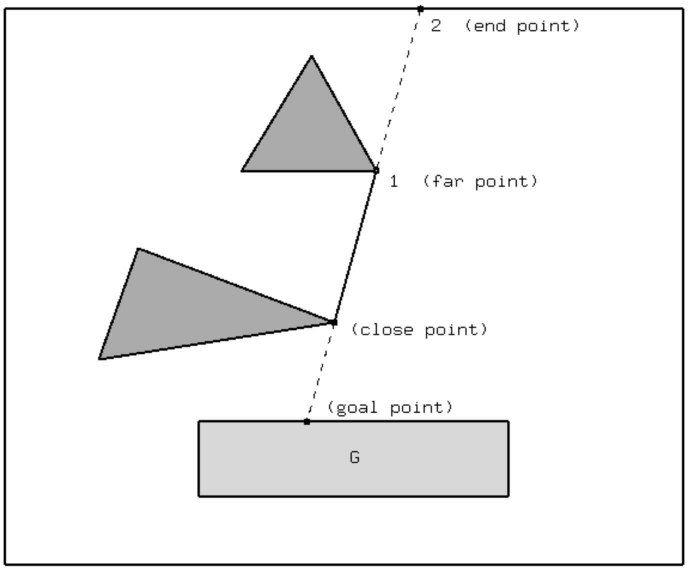

Figure 5.9: Determining the second point of the triangle: a case with the apex at the *far* point and an obstacle to the left. The ray can sweep counterclockwise around vertex 1.

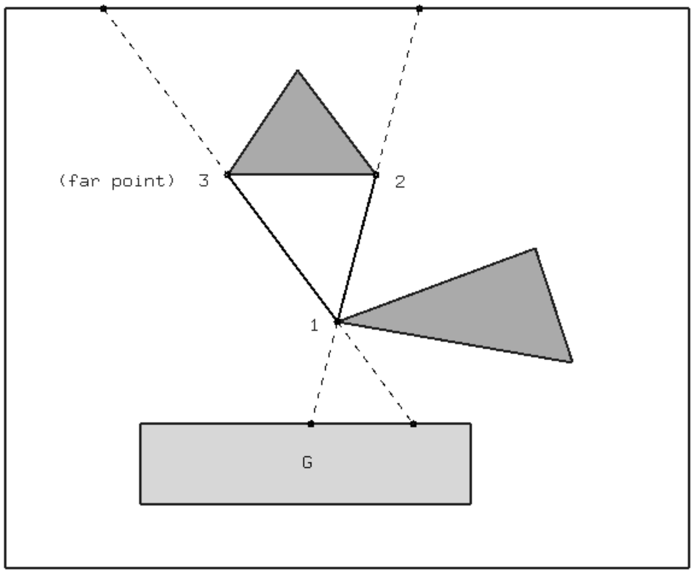

Figure 5.10: Determining the third point of the triangle: a case with point 2 at the *far* point of the first ray, and the *far* point of the second ray lies on the same obstacle edge.

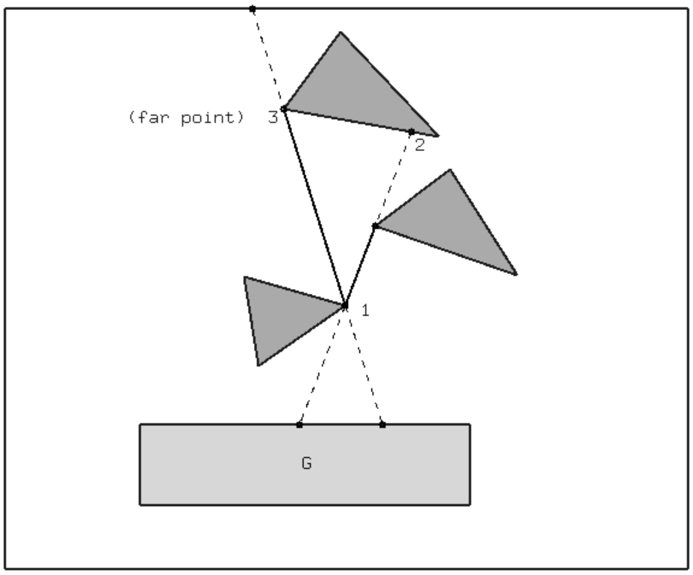

Figure 5.11: Determining the third point of the triangle: a case with point 2 at the *end* point of the first ray, and the *far* point of the second ray lies on the same obstacle edge.

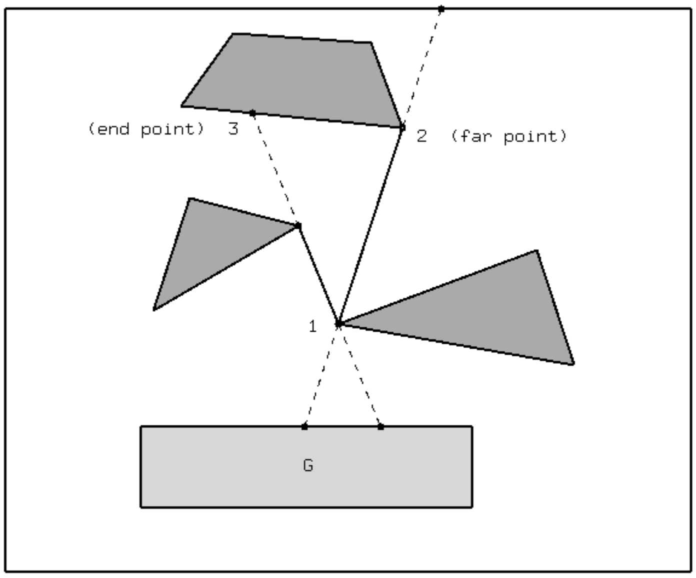

Figure 5.12: Determining the third point of the triangle: a case with point 2 at the *far* point of the first ray, and the *end* point of the second ray lies on the same obstacle edge.

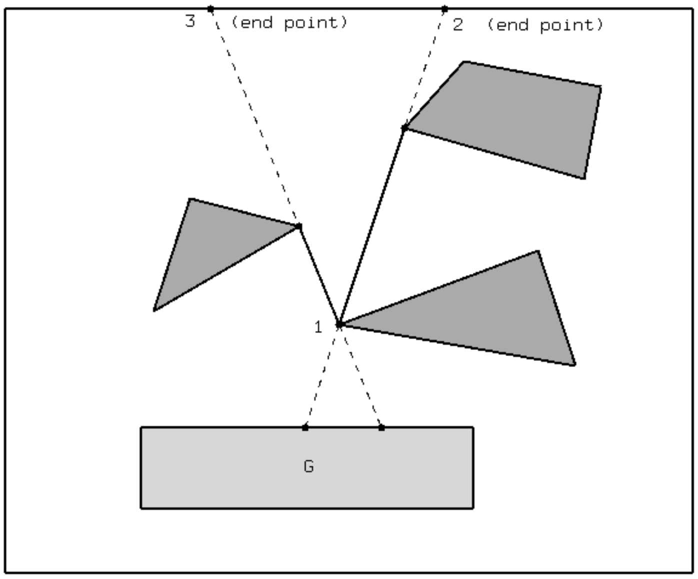

Figure 5.13: Determining the third point of the triangle: a case with point 2 at the *end* point of the first ray, and the *end* point of the second ray lies on the same obstacle edge.

In computing both upper and lower triangles, the determined endpoints are then checked for feasibility, *i.e.*, whether or not the swept triangle can be seen from the goal. Figures 5.14 and 5.15 show the triangles produced for two of the cases described above. Figure 5.16 shows an example in which two consecutive visibility rays do not sweep out a valid visibility triangle.

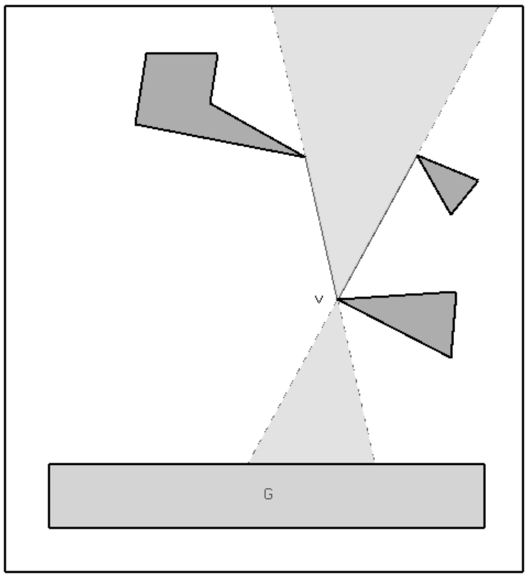

Figure 5.14: Upper and lower visibility triangles formed by two visibility rays at vertex v.

In the following pages we show the code for the routines described here.

The function "`visibility-triangles`" takes as input the extended visibility graph "`evg`", and all vertices and edges of the scene. It computes upper and lower visibility triangles at each vertex.

```
(define (visibility-triangles evg vertices edges)
  (cond ((null? vertices) '())
        (#t
         (let* ((vertex (car vertices))
                ; sort the visibility rays at the vertex by angle
                (rays (sorted-rays-at-vertex evg vertex)))
           (append (upper-triangles-at-vertex vertex rays edges)
                   (lower-triangles-at-vertex vertex rays edges)
                   (visibility-triangles evg (cdr vertices) edges))))))
```

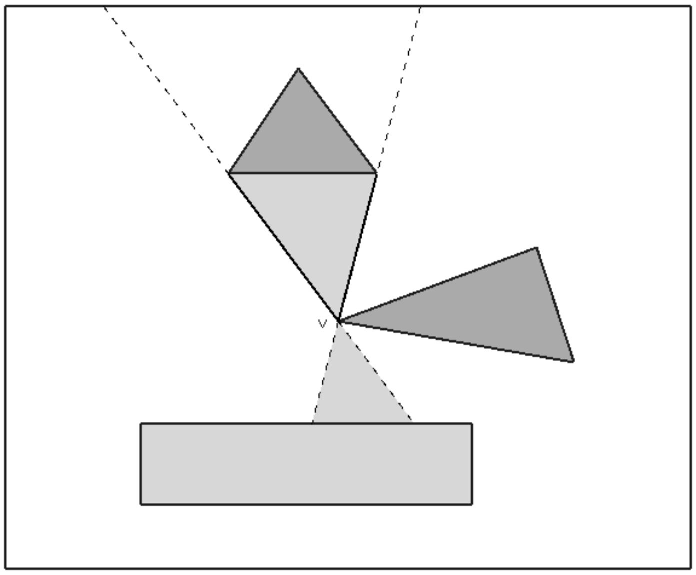

Figure 5.15: Upper and lower visibility triangles formed by two visibility rays at vertex v.

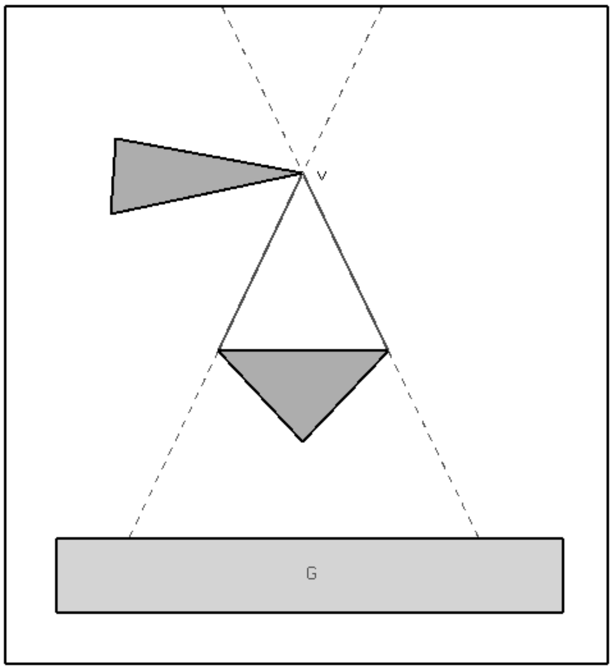

Figure 5.16: No visibility triangles are formed by these two visibility rays at vertex v.

```
(define (upper-triangles-at-vertex vertex rays edges)
  (cond ((null? rays) '())
        ((null? (cdr rays)) '())
        (#t (let* ((ray (car rays))
                   (next (cadr rays))
                   ; determine which polygon the close-pt of ray is on
                   (close-face (adjacent-face (ray->close-pt ray)
                                              (ray->vis-edge ray)))
                   ; determine second point of triangle
                   (second (if (and (close-pt? vertex ray)
                                    (ray-blocked? vertex ray))
                               (ray->far-pt ray)
                             (ray->end-pt ray)))
                   ; determine third point of triangle
                   (third (if (share-edge? second (ray->far-pt next) edges)
                              (ray->far-pt next)
                            (ray->end-pt next))))
              ; test whether [vertex, second, third] form a valid triangle
              (if (and (not (and
                                (edge-of-poly? (ray->edge ray) close-face)
                                (edge-of-poly? (ray->edge next) close-face)))
                       (not (and (ray-blocked? vertex ray)
                                 (far-pt? vertex ray)))
                       ; check that the vertices are oriented ccw
                       (= (triple-orientation vertex second third) *left*)
                       (share-edge? second third edges))
                  (cons (face-from-vertices (list vertex second third))
                        (upper-triangles-at-vertex vertex (cdr rays) edges))
                (upper-triangles-at-vertex vertex (cdr rays) edges))))))

(define (lower-triangles-at-vertex vertex rays edges)
  (cond ((null? rays) '())
        ((null? (cdr rays)) '())
        (#t (let* ((ray (car rays))
                   (next (cadr rays))
                   (second (ray->goal-pt ray))
                   (third (ray->goal-pt next)))
              ; test whether [vertex, second, third] form a valid triangle
              (if (and (= (triple-orientation vertex second third) *left*)
                       (share-edge? second third edges)
                       (not (and (ray-blocked? vertex ray)
                                 (far-pt? vertex ray))))
                  (cons (face-from-vertices (list vertex second third))
                        (lower-triangles-at-vertex vertex (cdr rays) edges))
                (lower-triangles-at-vertex vertex (cdr rays) edges))))))
```

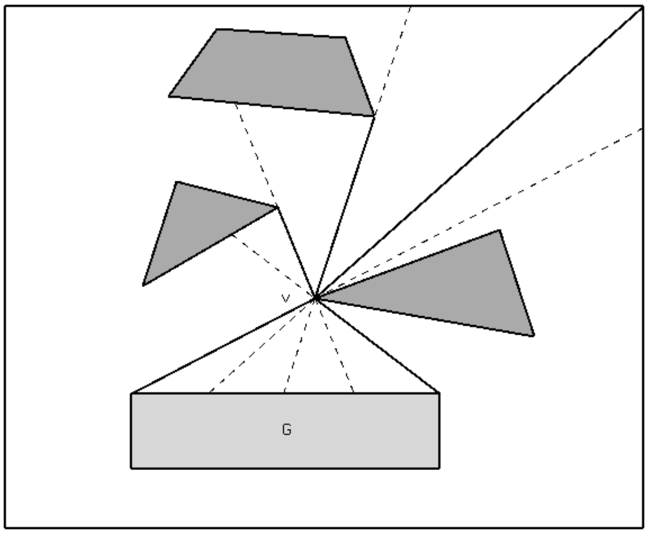

Figure 5.17: All visibility rays at vertex v.

The union of all the valid triangles forms the partial visibility region. Our system simply computes a list of the triangles and draws them in the graphics window rather than explicitly computing their union. Figure 5.17 shows all the visibility rays anchored at a single vertex v. Figure 5.18 shows all the visibility triangles swept out by these rays.

Below is a sample driver for the partial visibility system.

```
(define (partial-visibility . scene)
  (let ((scene (if (null? scene) (get-scene) (car scene))))
    (redraw scene)
    (let* ((polys (cons (bbox scene)
                        (cons (goal scene) (obstacles scene))))
           (edges (polys->edges polys))
           (vertices (polys->vertices polys))
           (vg (vgraph polys))
           (evg (extended-vg vg scene))
           (tris (visibility-triangles evg vertices edges)))
      (display *cr*)                        ; carriage return
      (display " Extended visibility edges: ") (display *cr*)
      (draw-rays evg)
      (mouse-click-pause
       " Click anywhere to see partial visibility regions: ")
      (draw-partial-vis scene tris)
      (overlay-scene scene)
      (mouse-click-pause
       " Click again to see shadow regions under partial visibility: ")
      (draw-vis+shadows scene tris))))
```

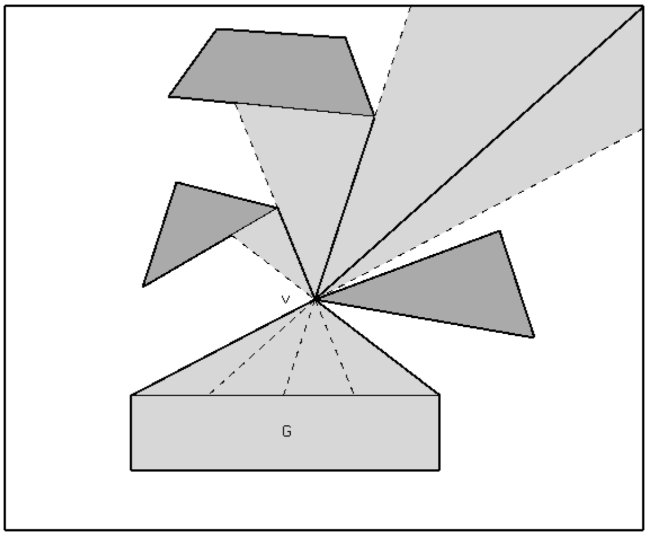

Figure 5.18: All visibility triangles at vertex v.

The following figures illustrate the components of the partial visibility system described above. In Figure 5.19 we show an example scene in which the goal region is shaded lighter than the obstacles. Figures 5.20 and 5.21 give the visibility graph and extended visibility graph, respectively, for this scene. Figure 5.22 shows a few of the visibility triangles, and Figure 5.23 shows the partial visibility region (the union of all the visibility triangles).

5.1.2 Implementation of the complete visibility algorithm

To compute complete visibility regions, we employ the plane sweep code provided by the LIMITED system [Don89]. The first step of our algorithm is to compute the inner tangents between each obstacle and the goal. For a given obstacle O, the tangents are computed by first computing the complete graph between the vertices of O and the vertices of the goal. Each edge of the complete graph gets extended in both directions so that its endpoints lie outside the bounding polygon. If the extended edge is not a local tangent of both the obstacle O and the goal, it is immediately discarded. Otherwise the visibility edge is checked to ensure that obstacle O and the goal lie on opposite sides of it, and to determine if the visibility edge intersects the interior of an obstacle or the goal. A visibility edge that passes these tests extends from the goal to some point on an edge in the scene. We call these extended edges *tangents*.

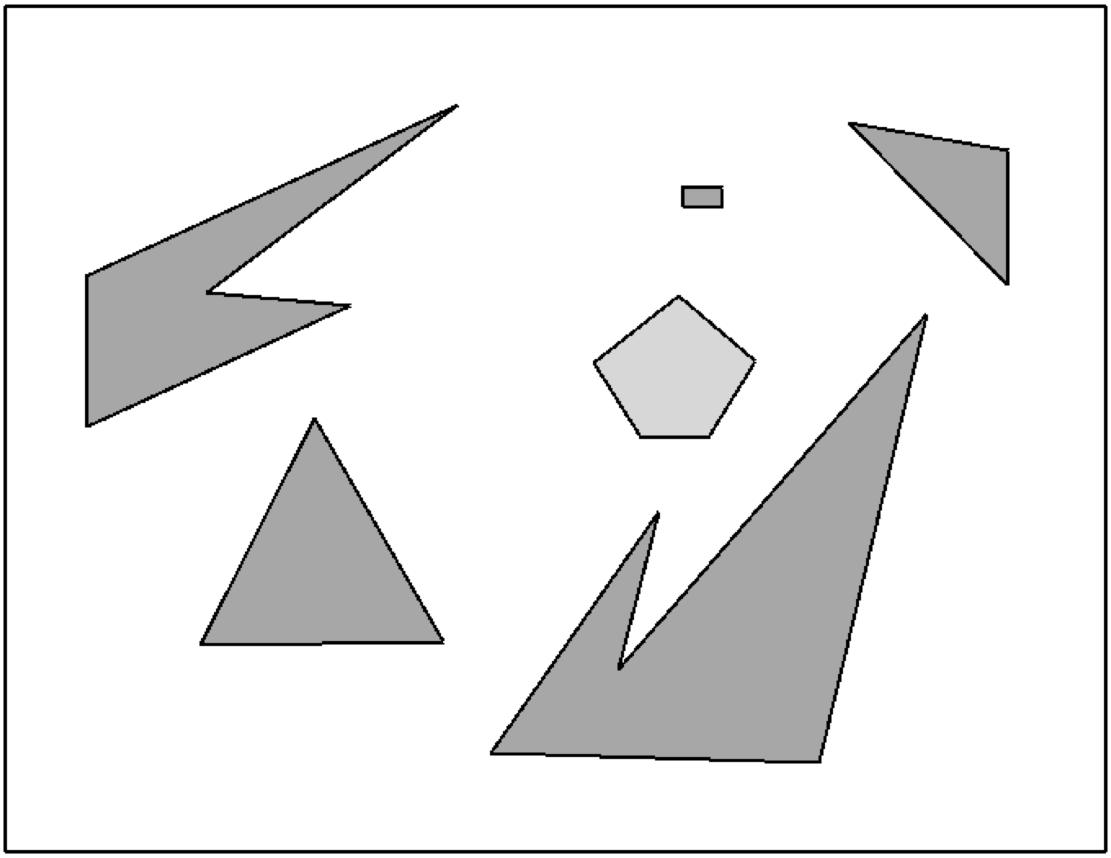

Figure 5.19: An example scene.

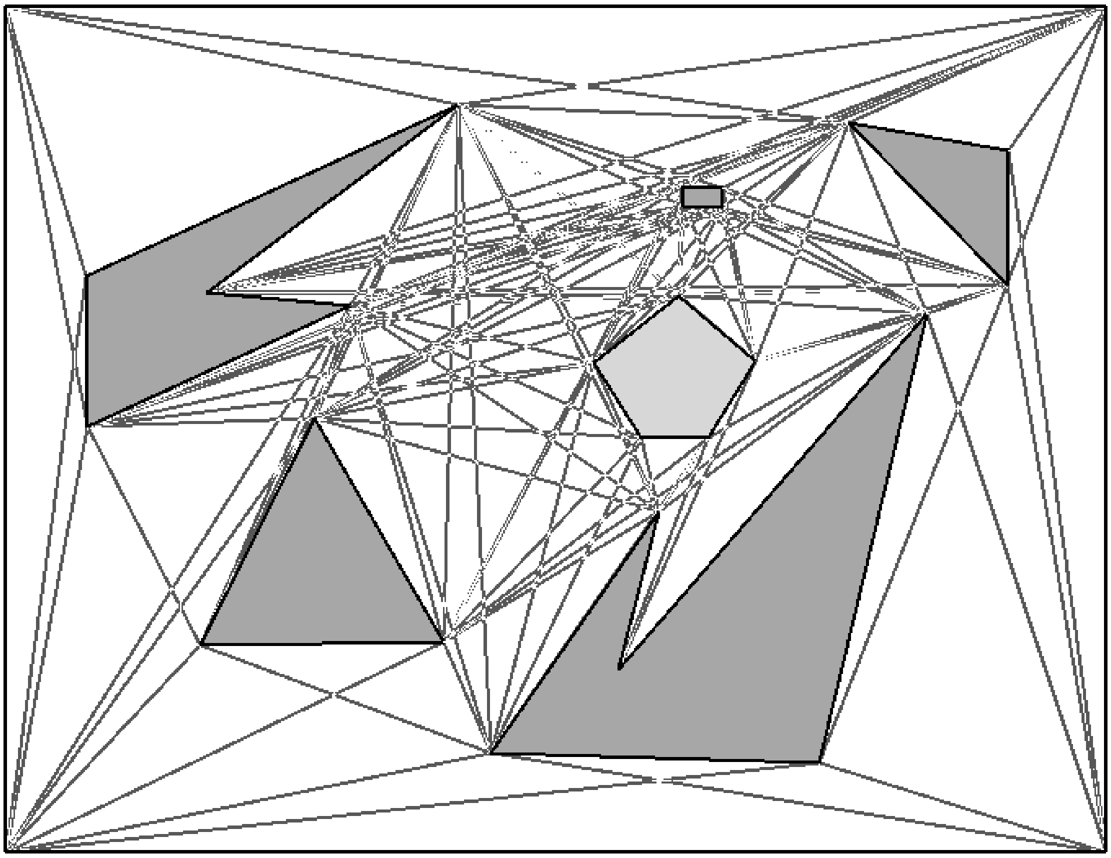

Figure 5.20: The visibility graph.

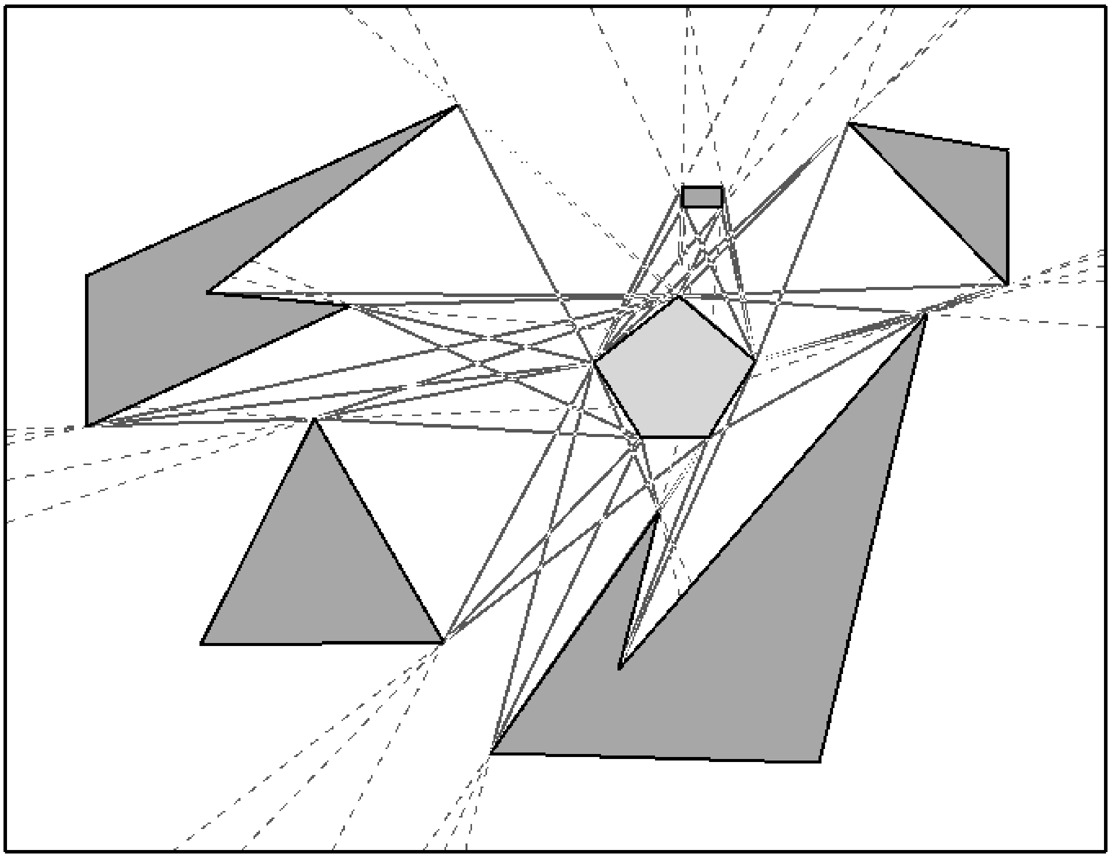

Figure 5.21: The extended visibility graph.

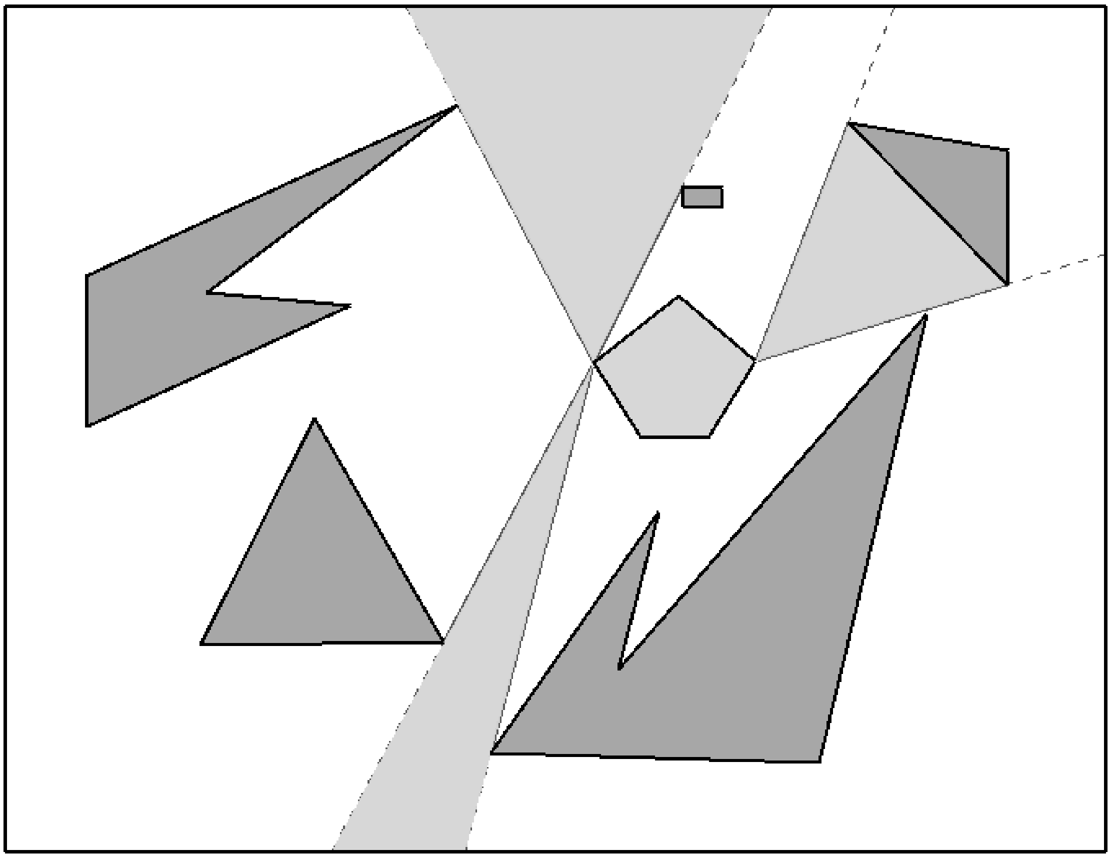

Figure 5.22: A few visibility triangles for this example.

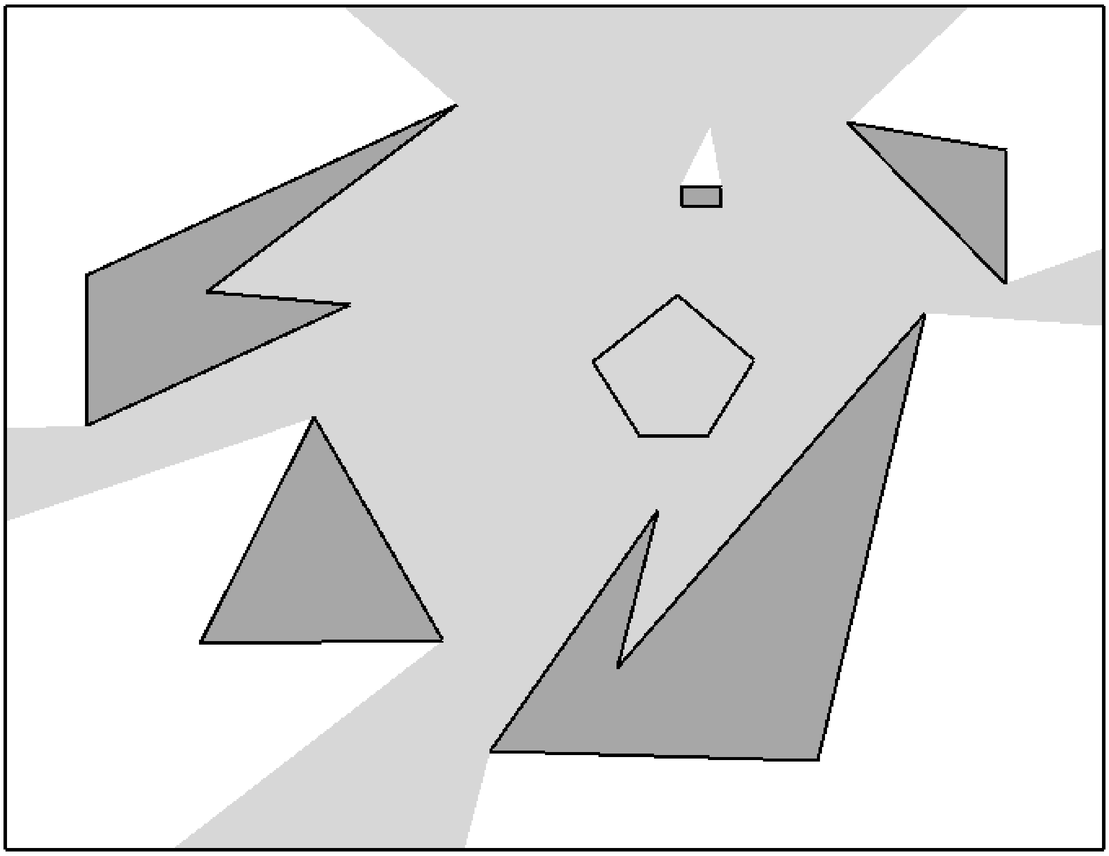

Figure 5.23: The partial visibility region.

Each tangent is represented as a list of the following components:

1. the endpoint on the goal
2. the point of tangency with a vertex (obstacle or bounding polygon) in the scene
3. an endpoint in the environment (on an obstacle or the bounding polygon).

Below is the top-level routine from the complete visibility system that computes all local inner tangents between the goal and a given obstacle.

```
(define (inner-tangents poly goal obstacles bbox)
  (let loop ((edges (complete-graph poly goal))
             (ts '()))
       (cond ((null? edges) ts)
             (#t
        (let* ((vedge (orient-from-poly (car edges) goal))
               (extended (extend vedge *scale-factor*))
               (goal-pt (edge-tail vedge))
               (dark-pt (edge-head vedge))
               (front (make-edge dark-pt
                                 (edge-head extended))))
          (if (and (local-tangent-at-vertex? extended
                                             dark-pt
                                             poly)
                   (local-tangent-at-vertex? extended
                                             goal-pt
                                             goal)
                   (not (poly-intersect? extended goal *no-endpts*))
                   (not (sameside?
                         (next-vertex-of-poly dark-pt poly)
                         (next-vertex-of-poly goal-pt goal)
                         vedge)))
              (let* ((end-pt (find-endpt front obstacles bbox))
                     (tangent (list goal-pt
                                    dark-pt
                                    end-pt)))
                (loop (cdr edges) (cons tangent ts)))
                      (loop (cdr edges) ts)))))))
```

Once we have the tangents, we create a list of *tangent rays* containing the part of each tangent starting at the point of adjacency with an obstacle vertex and extending away from the goal. We then compute the arrangement of the scene with this list of tangent rays. This is a bit tricky, since the arrangement code we use in the system computes arrangements of polygons, not lines or edges. We create long thin polygons to represent the rays, and keep track with a hash table of how the rays and polygons correspond to each other.[1] The arrangement procedure returns the arrangement in the form of a list of polygons, along with a log-time query tree. Given the arrangement, the complete visibility region of the goal corresponds to the cell that contains the goal. We locate this cell by seaching the query tree that is returned with the arrangement, as shown here:

```
(define (compute-light-region polygon AQ)
  (let* ((pt (edge-head (car (boundary polygon))))
         (L-poly (car (locate-pt-cell pt (cadr AQ) epsilon)))
         (J-poly (limited-polygon->JAR-polygon L-poly)))
    J-poly))
```

Recall that our system relies on Jonathan Rees' geometric modeling system, Bruce Donald's plane sweep module LIMITED, and code to tie these systems together. The following points are mentioned to clarify the above code:

- The *boundary* of a cell is a set of other cells of lower dimension. A vertex has no boundary, an edge's boundary consists of two vertices, and a face's boundary is a set of edges.
- The function `locate-pt-cell` takes as input a point, a query-tree, and an ϵ to be used for converting from our representation of points to the LIMITED representation. It returns the arrangement cell containing the given point.
- The function `limited-polygon->JAR-polygon` converts a LIMITED polygon to the polygon representation in Jonathan Rees' system. Since the latter system does not handle polygons with holes, holes in the LIMITED polygon are ignored.

Figure 5.24 shows the tangents for the scene of Figure 5.19. Figure 5.25 shows the complete visibility region for this scene.

[1] The representations of geometric objects in Bruce Donald's plane sweep system (written in Common Lisp) is different from that of Jonathan Rees' geometric modeling system (written in Scheme). We have code to convert between these representations and tie these systems together.

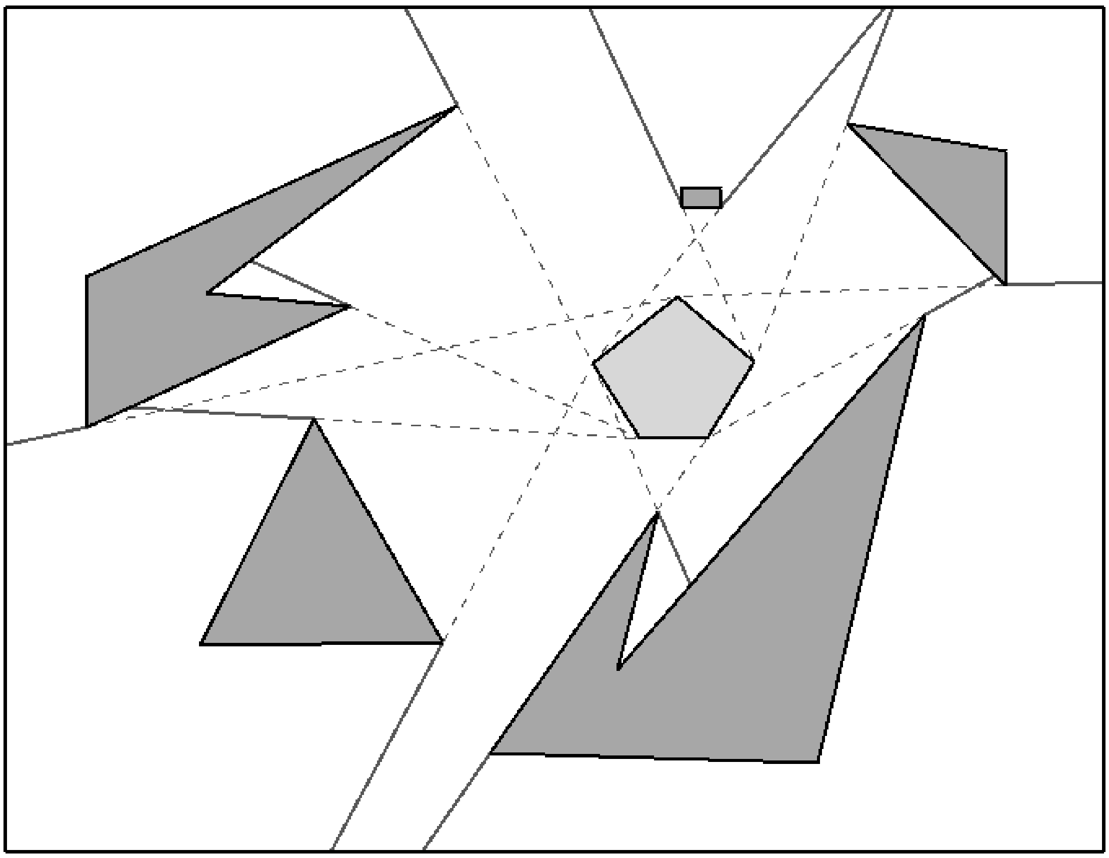

Figure 5.24: The inner tangents for the scene of Figure 5.19.

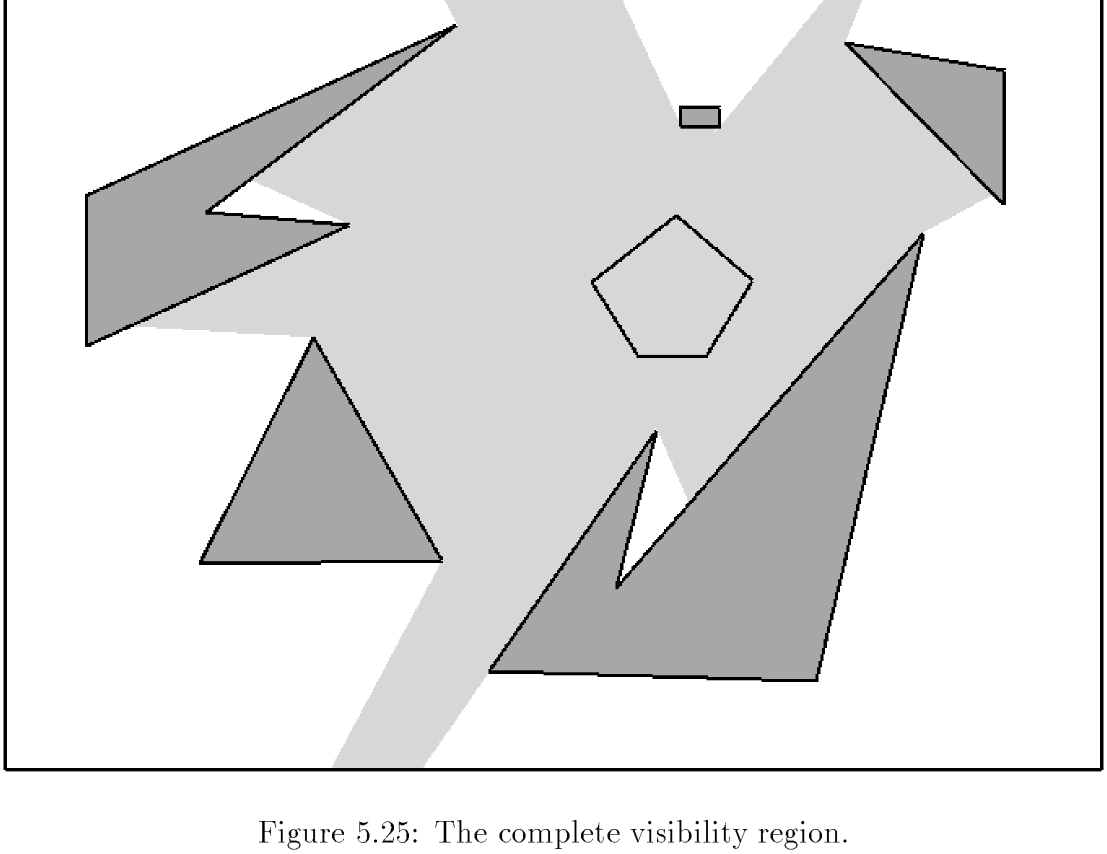

Figure 5.25: The complete visibility region.

Below is a sample driver for the complete visibility system.

```
(define (complete-visibility . scene)
  (let ((scene (if (null? scene) (get-scene) (car scene))))
    (redraw scene)
    (display " Computing tangents...") (display *cr*)
    (let ((tangents (compute-tangents scene)))
      (display " Computing arrangement...") (display *cr*)
      (let ((AQ (compute-arrangement scene tangents epsilon)))
        (display " Computing complete visibility region...")
        (display *cr*)                   ; carriage return
        (let ((cell (compute-light-region (goal scene) AQ)))
          (redraw scene)
          (display " Complete visibility region: ") (display *cr*)
          (draw-vis-regions (list cell))
          (overlay-scene scene)
          (mouse-click-pause " Click to see shadow regions: ")
          (draw-shadows scene (list cell))
          (showit))))))
```

5.2 The robot surveillance project

We used the implementation of the complete visibility algorithm to build a demonstration of robot surveillance using two of the mobile robots in the Cornell Robotics and Vision Laboratory. Figure 5.26 shows a picture of the two autonomous mobile robots we used in the demo. Their names are TOMMY and LILY. The task was for TOMMY to detect when LILY entered a particular doorway of the robotics lab. Initially TOMMY is at a position from which this doorway cannot be seen. Below we describe the various components of the system.

5.2.1 A note on the lab and our robots

The Cornell Robotics and Vision Laboratory was built and is supervised by Bruce Donald and Dan Huttenlocher. Over the past few years, several general-purpose autonomous mobile robots have been designed and built in the lab by a team of student researchers led by Russell Brown and Jim Jennings [Bro95,Jen95]. They have been used for teaching several undergraduate and graduate robotics and vision classes, as well as for many research projects, including work on cooperating robots [DJR93a,DJR93b,DJR94], manipulation [JR93,Jen95], map-making and robot localization [BCD93,Bro95], task-directed planning [DJ91,DJ92a], and visually-guided navigation [HLR94]. Much of the hardware has been specifically designed for the robots in our lab. This has included the development of the Cornell Generic Controller that connects the robot CPU with on-board devices such

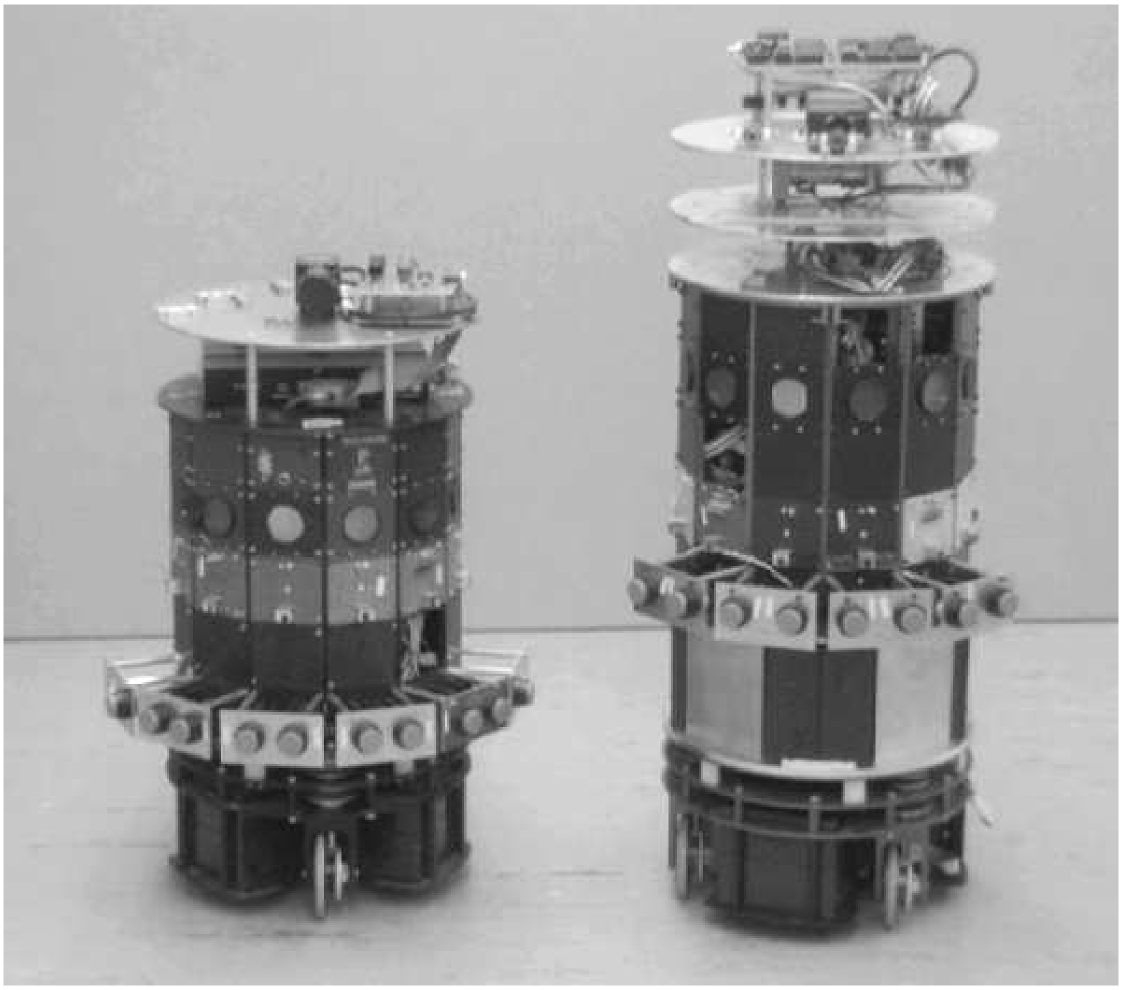

Figure 5.26: The mobile robots TOMMY and LILY. TOMMY is on the left, LILY is on the right.

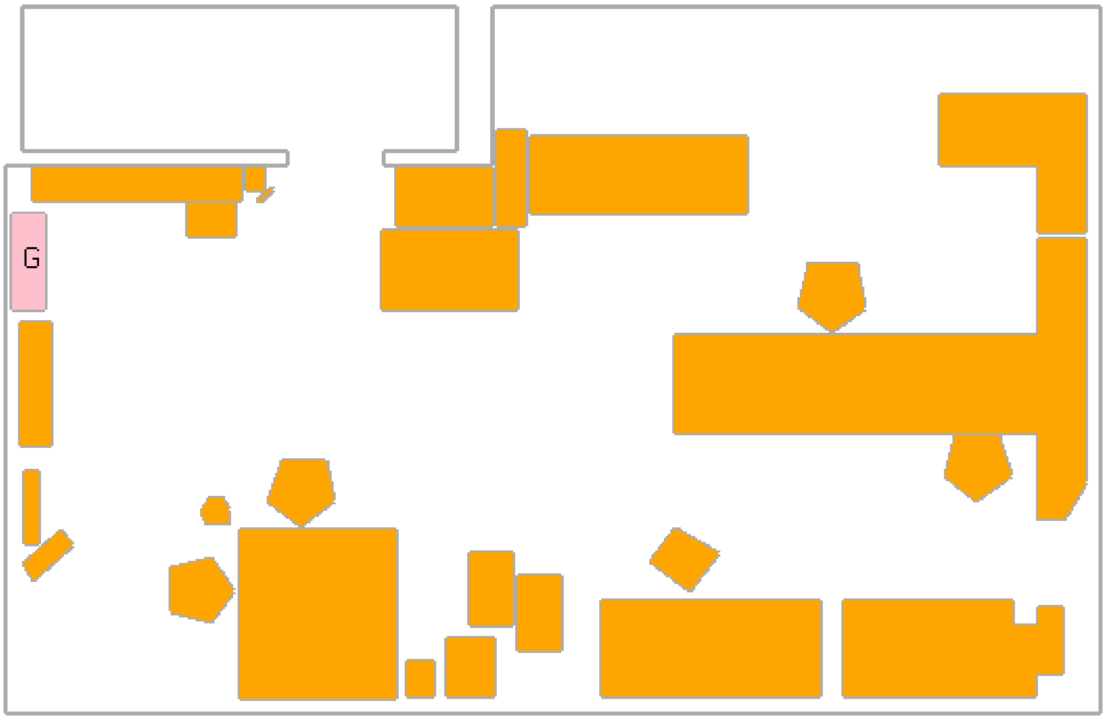

Figure 5.27: Our map of the robotics lab.

as sonar, bump sensors, infrared sensors, and a laser range finder. The robots are built on top of RWI wheelbases. Users can program the robots in Scheme using an interactive, multi-thread Scheme interpreter designed and implemented by Jonathan Rees [RD92].

5.2.2 The visibility component

We constructed by hand a map of our lab, and used that map as the input scene to the complete visibility system described above. The map is shown in Figure 5.27.

TOMMY's task was to monitor the doorway, which is marked in the Figure with "G". The dark gray regions are obstacles representing real objects in the room — chairs, desks, couches, bookshelves, *etc.* Given that most of the objects are regularly shaped and resting on the floor, the idea of using polygons as "footprints" of $3D$ objects turned out to give a good approximation of the $3D$ geometry. Given this map, our algorithms give us the exact placements from where the doorway can be monitored. The lightly shaded region in Figure 5.28 is the complete visibility region for this scene — the exact set of placements from where the doorway can be entirely seen with a sensing device such as a CCD camera.

Figure 5.28: The complete visibility region for the robotics lab.

5.2.3 Choosing a new placement

Based on the visibility region and the initial configuration of TOMMY, a new configuration is computed inside the visibility region. A motion plan to reach that new configuration is generated along with the distance from there to the goal.

In particular, we do the following to choose such a placement. We first shrink the visibility region to account for uncertainty. The procedure to perform this shrinking returns a list of edges making up the *shrunk visibility region*. We now want to choose a new point inside this shrunk visibility region, one that is closest to the current position of the robot. We use the following heuristic to find such a point: we discretize the edges of the shrunk visibility region, obtaining a list of candidate points. We then sort this list of points by distance from the current position of the robot. Then test each of the points, searching for one that is reachable from the current position in a one-step motion. The first such point found is returned as the new configuration. If no such point is found, this is signalled. This could be due to two reasons: a point reachable in a one-step motion was missed due to the discretization being too coarse, or no one-step motion plan exists (*i.e.*, the mobot would have to move around corners, or cannot reach the visibility region at all). While the former case could easily be fixed by iteratively refining the discretization, the latter case requires the use of a full-fledged motion planner (*e.g.*, as described

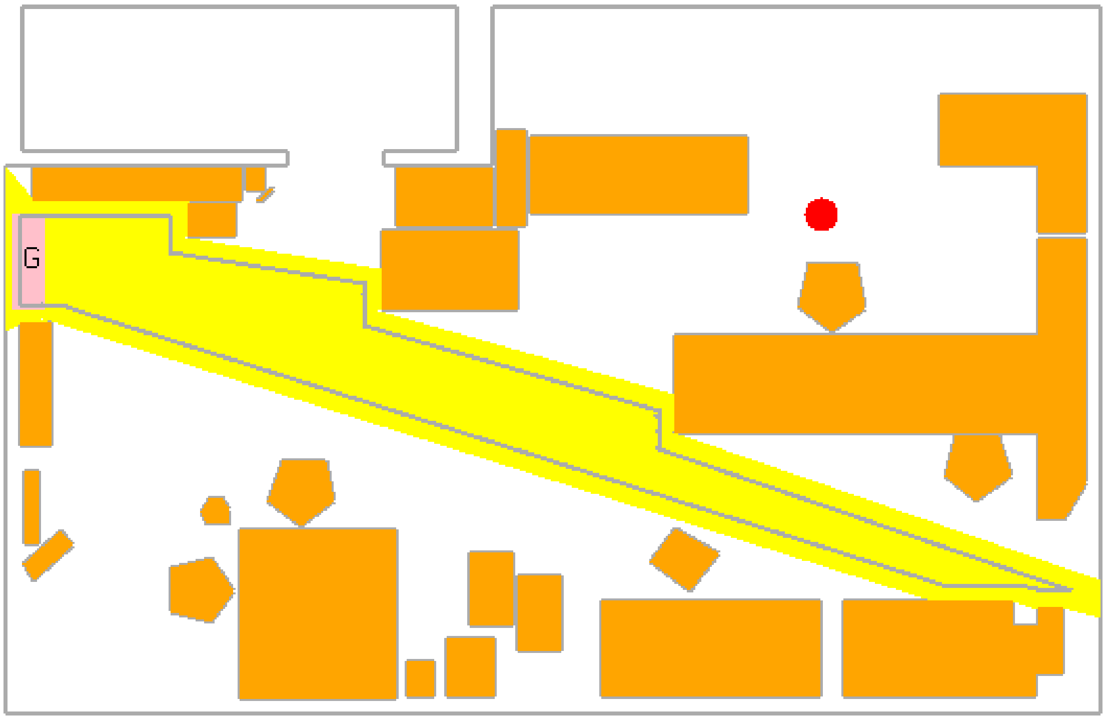

Figure 5.29: The shrunk visibility region and an example starting configuration of TOMMY in the robotics lab.

in [Buc87,Don87,Lat91]). This would be overkill for this project.

Figure 5.29 shows the shrunk visibility region and one of the starting points we used, and Figure 5.30 shows the new placement which was computed using the method described above.

5.2.4 Computing the viewing direction

The planner computes a viewing direction depending on the new placement and information obtained from the map. We fix a coordinate frame for the lab with the origin in one corner of the room. Then we compute the vector between the new computed placement and the centroid of the goal. The θ component of the new configuration is simply the angle of this vector. The final output from the planner is a vector containing the x-, y-, and θ-components of the new configuration, along with the distance in world coordinates from this new configuration to the centroid of the goal.

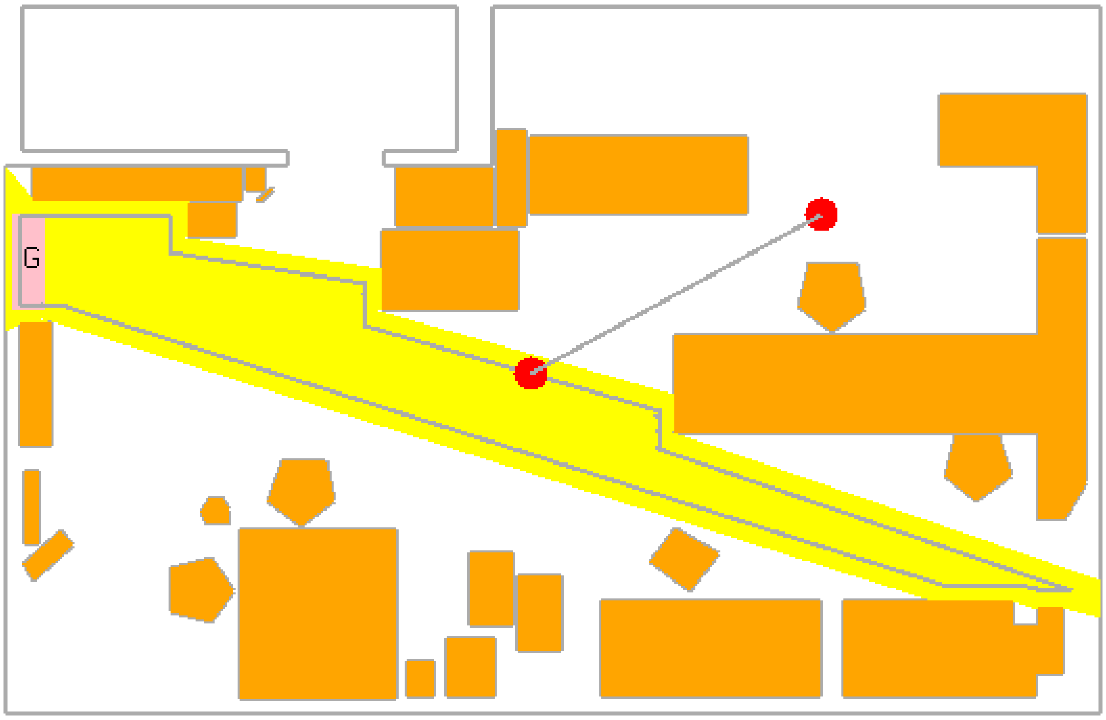

Figure 5.30: The computed new configuration and the one-step motion.

5.2.5 LILY's job

LILY has the simple task of moving into the doorway and waiting for TOMMY. LILY is run without a tether. She is programmed to translate a fixed distance and stop (in the center of the doorway). She then waits until her bump sensors are activated. When a bumper is pressed, LILY translates a fixed distance in reverse, rotates by 180 degrees, and then translates forward a fixed distance in order to leave the room.

5.2.6 Locating LILY

Here is how the surveillance and recognition parts of the system work.

We first built a calibrated visual model of LILY. We used the Panasonic CCD camera mounted on TOMMY to take a picture of LILY from a known fixed distance (4 m). We then computed the intensity edges for that image using an implementation of Canny's edge detection algorithm [Can86]. The actual model of LILY that we created and used is shown in Figure 5.31. We did not alter the intensity edges that Canny's algorithm output, and we feel that our results are relatively insensitive to the particular image we took.

Based on the distance information from TOMMY's new configuration, the model edges are scaled to the expected size of LILY's image as seen from this configuration, using the fact that the image size is inversely proportional to the distance.

The video camera on TOMMY is used to repeatedly grab image frames, which along with the scaled model are input to a matcher that operates on edge images. The following loop is performed until a match is found:

1. Grab a frame.
2. Crop it, keeping only the portion of the image where LILY is expected to be.
3. Compute intensity edges for the cropped image.
4. Run the matcher to find an instance of the scaled model in the cropped image.

Figure 5.32 shows one of the images that was grabbed with TOMMY's videocamera once TOMMY had moved into the computed configuration. Figure 5.33 shows the intensity edges for the cropped image of Figure 5.32.

The matcher used in the experiment is based on the Hausdorff distance between sets of points and was written by William Rucklidge [Ruc95]. It is based on work by Huttenlocher and Rucklidge [HR93], and has been used extensively in the Cornell Robotics and Vision Laboratory for image comparison, motion tracking, and visually-guided navigation [HKR93,HJ94,HLR94]. For a definition of the Hausdorff distance, see Appendix A. Extensions to the basic definition have been developed to deal with missing data, gross outliers, and small perturbations in the locations of the points [HKR93,HJ94]. The particular matcher used here is a translation-only matcher that uses a fractional measure of the Hausdorff distance. Matches are found by searching the $2D$ space of translations of the model, and computing the Hausdorff distance between the image and the translated model. A match occurs when the Hausdorff distance of a certain fraction of the points is below some specified threshold. All translations of the model that fit the image are returned.

The matcher allows the user to specify what fraction of the points should be matched in both the forward and reverse directions. The *forward* threshold spec-

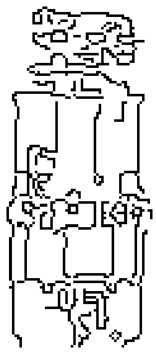

Figure 5.31: Our model of LILY.

Figure 5.32: An image grabbed during the experiment.

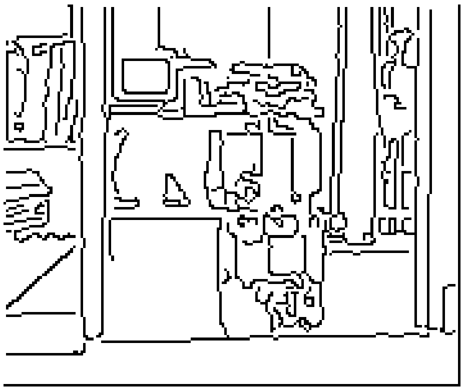

Figure 5.33: Intensity edges for the cropped image.

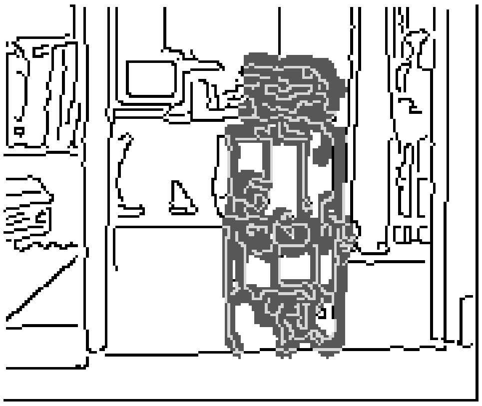

Figure 5.34: Matches found between the image and the scaled model.

ifies what fraction of the image points must lie within a specified distance of a model point. The *reverse* threshold specifies what fraction of the model points must lie within a specified distance of an image point. We called the matcher with a forward threshold of .8 within a 2-pixel radius, and a reverse threshold of .65 within a $\sqrt{2}$-pixel radius. These settings of the parameters were experimentally determined. They were low enough that at least one match was always found, even when lighting conditions varied significantly. Usually, many matches were found (all within ϵ of the true match). No false matches were ever found with these settings of the parameters.

The dark gray outline in Figure 5.34 shows all matches that were found between the scaled model of LILY and the image in Figure 5.33.

5.2.7 The final action

Once TOMMY has recognized LILY, the final action of the system is that TOMMY moves to where LILY is, and then the two mobots leave the lab together. Based on where LILY is found in the image, TOMMY first performs a rotational correction so that LILY is centered in the image. This rotation is performed to compensate for errors in dead reckoning. Computing this correction requires that we know the focal length f of the camera.

To determine f experimentally, we took a picture of an object with known size x at known distance d. If p is the apparent size of the object in the image, we can compute f using the following equation:

$$\frac{x}{d} = \frac{p}{f}$$

where d and x are measured in millimeters and f and p are measured in pixels. See Figure 5.35 for an illustration of this correspondence. If LILY is found p pixels

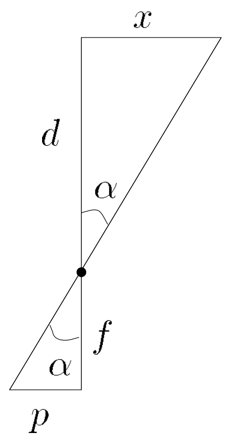

Figure 5.35: Determining the camera focal length.

off center in the image, TOMMY has to rotate by an angle α, where

$$\tan \alpha = \frac{p}{f}$$

or

$$\alpha = \arctan \frac{p}{f}.$$

For small angles, we can approximate

$$\alpha \approx \frac{p}{f} = ap$$

where

$$a = \frac{180}{\pi f}$$

is the number of degrees per pixel. We obtained $a = .0804$ experimentally.

Using this value, TOMMY performs a rotational correction as seen in the following code fragment:

```
(let ((correction-angle (goto-and-match-model (new-config)
                                               (goal-distance))))
  (cond (correction-angle
         (rotate 'by correction-angle)
         (goto-lily))
```

Once LILY is centered in the image, TOMMY moves across the room to where LILY is. The actual command is a simple guarded move:

```
(tolerate-impediments
    (translate 'by (- (goal-distance) mobot-radius)))
```

When TOMMY and LILY collide, TOMMY senses the impediment and says "Hi Lily" using an on-board speech synthesizer. At the same time, LILY's bump sensors get activated. LILY backs up, turns, and leaves the room. TOMMY waits briefly, rotates to a desired heading to face out the door, and then translates by a fixed amount through the doorway to follow. Alternatively, we could have used TOMMY's "heel" command [RD92] to have TOMMY follow LILY using sonar.

5.2.8 An overall view of the system

Figure 5.36 shows a block diagram of all the components of the system.

A behavioral description of the system is on the left:

1. A new configuration is computed.
2. A motion plan to move to that new configuration is executed.
3. When the motion is completed, the surveillance system starts up: grabbing an image, cropping, and edge detecting.
4. Image edges and scaled model edges are sent to the Hausdorff matcher.
5. If a match is found, the location of the match is used to guide a motion of TOMMY; if no match is found, another image is grabbed.

We videotaped several runs of the system. For these runs, we used two different starting positions for TOMMY, on different sides of the room, both from where the goal doorway could not be seen. We also demonstrated the robustness of the system by having people enter and leave through the doorway while TOMMY was monitoring it. The system performed consistently well. TOMMY never reported a false match — neither when the doorway was empty, nor when other people stood in the doorway. Once LILY was in position, the recognition component typically took about 8 seconds to locate LILY. Disk access time accounted for some of this time (saving and loading image files) and could be eliminated by using a different file access strategy.

The code loaded onto TOMMY is given in Appendix B.

5.3 Discussion

In this chapter, we described the systems contributions of this thesis. The complete and partial visibility algorithms for a stationary robot (presented in Sections 3.3.1 and 3.4.1, respectively) have both been implemented on top of existing packages for graphics and geometric modeling. In addition, we described our robot surveillance demonstration system, which was built using the complete visibility algorithm to compute sensor placements. We feel that the success and reliability of this work demonstrates the validity of developing geometric algorithms and systems for the planning of robot sensing and control strategies.

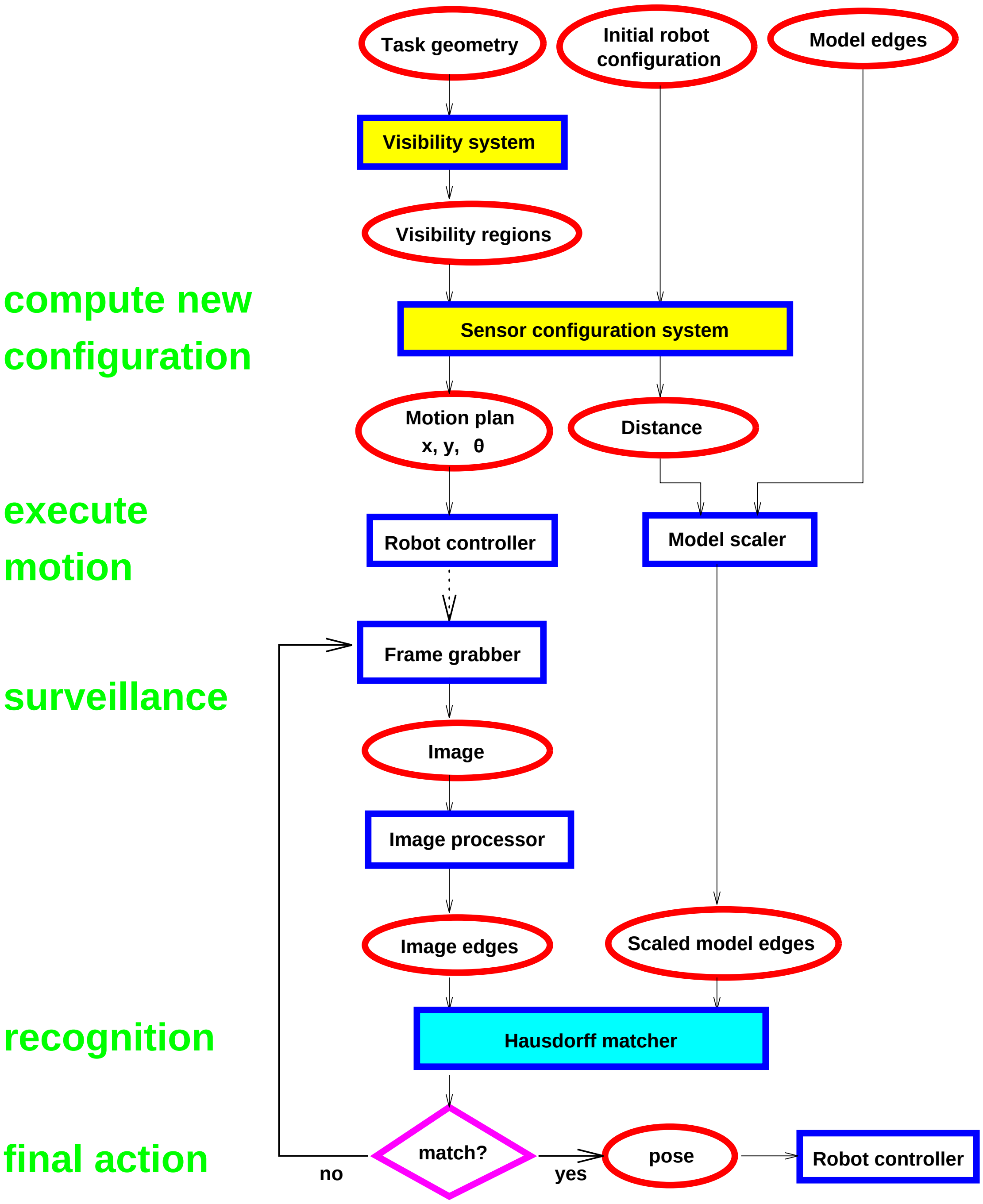

Figure 5.36: Block diagram of the robot surveillance system.

Chapter 6

Conclusion

In this thesis, we considered several challenging problems in robotics and gave efficient algorithms for solving tasks in motion planning and sensor configuration planning.

6.1 Motion planning

In our motion planning work, we gave an $O(n^2 \log n)$ algorithm for finding a commanded motion direction θ that guarantees a trajectory from a specified start region to a specified goal region amidst planar polygonal obstacles where control is subject to uncertainty. This result represents a quadratic improvement over the best previous bounds. It was achieved by a new analysis of the geometric complexity of the non-directional backprojection. This careful analysis yielded an efficient algorithm for computing a representation for the non-directional backprojection.

We expect that an implementation of this algorithm would perform well. Our robotics lab at Cornell has implemented an approximate generalized damper on a PUMA 560 robot and found through experiments that the control error cones are small enough that implemented backprojection algorithms perform well [JDC89].

6.2 Sensor configuration planning

In studying automatic sensor configuration planning, part of our motivation was to shed light on the complexity of recognizing goal attainment. We have given algorithms for computing goal recognizability with an idealized sensor under two notions of detectability. Complete visibility of the robot in the goal can be computed efficiently, and provides a good model of detectability in an uncluttered environment. In a complicated environment, the weak visibility model affords more sensor configurations from which the robot may be partially visible. We have also described how the regions we compute can be restricted to handle errors in sensor configuration. In particular, we considered errors in sensor placement and aim. A surprising result is that the consideration of directional uncertainty lowers the complexity of computing recognizability regions.

6.3 Systems and experiments

We have implemented our algorithms for computing visibility regions for a static polygon in both the complete and partial visibility models. The implementations were very valuable in elucidating the underlying structure of visibility regions. The implementation of the complete visibility algorithm was used as part of a demonstration system of robot surveillance using two of the mobile robots in our lab. This system demonstrated both the robustness and applicability of the visibility algorithms we have developed. Furthermore, we feel that this successful effort has validated our principled approach to planning robot sensing and control strategies.

6.4 Future directions

Within the partial visibility model, we have presented algorithms for computing visibility maps for a stationary robot and for a robot translating in a goal region. We plan to extend this work to compute partial visibility maps for a translating and rotating robot. This would enable partial visibility of a robot at any position and orientation within a goal region.

We plan to extend the uncertainty model to handle sensor reading errors in addition to sensor configuration errors. Another extension is the computation of efficiently-computable conservative approximations to recognizability regions.

We believe that the techniques of Section 3.3 apply to the case of computing complete visibility regions in $3D$. Our conjecture is that a direct application of our techniques yields an algorithm of running time $O(n^2\alpha(n))$, for a convex feature polygon of size m and a polyhedral environment of complexity n. This would improve the best previous bound of $O(N^{2+\delta^2})$ for $N = O(mn)$ by Tarabanis and Tsai [TT92].

We are currently working on extensions to the work presented in Section 3.5. In particular, we expect our work on the fat triangles model of visibility to have broad applicability, both in sensor planning and in motion planning without compliance.

We look forward to pursuing the connection between our work and the information invariants theory of Donald [Don93a,Don93b].

In the systems area, we plan to implement our methods for incorporating sensor uncertainty, and to experiment with using partial visibility maps in robot surveillance. We also plan to apply our techniques to more applications in the area of active sensing.

Appendix A

The Hausdorff distance

Given finite point sets A and B in $\mathbb{R}^2$, the Hausdorff distance between A and B is defined to be the maximum of the directed distances, *i.e.*,

$$H(A, B) \quad = \quad \max(h(A, B), h(B, A))$$

where the directed distance is

$$h(A, B) \quad = \quad \max_{a \in A} \min_{b \in B} d(a, b)$$

and $d(a, b)$ is the Euclidean distance between points a and b.

Figure A.1 shows an illustration of this definition. In computing the Hausdorff distance between edge images, we consider each pixel to be a point. See Rucklidge's Ph.D. dissertation for an excellent discussion of the Hausdorff distance and its use in visual recognition [Ruc95].

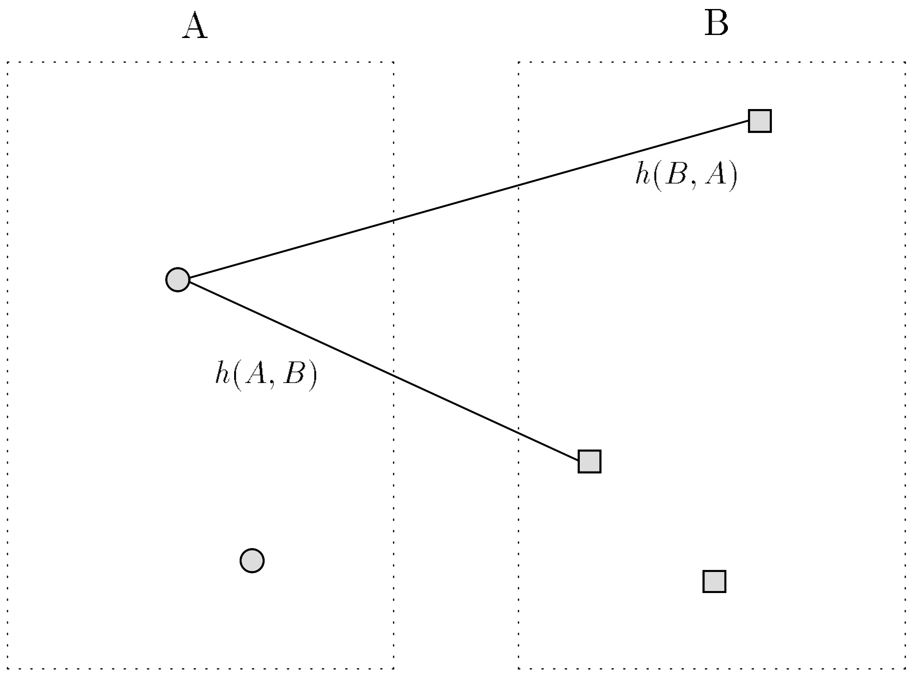

Figure A.1: An illustration of the Hausdorff distance. In this example, the Hausdorff distance is $H(A, B) = h(B, A)$.

Appendix B

Code from the robot surveillance system

Our robot surveillance system is described in Section 5.2. It was built using the complete visibility system described in Section 5.1.2, and the Hausdorff matcher by William Rucklidge [HR93,Ruc95].

In Section B.1 we give specifications for some of the mobile robot routines we use. Section B.2 gives the code that was loaded onto the mobile robot TOMMY.

B.1 Mobot Scheme primitives

Researchers Jonathan Rees, Jim Jennings, and Russell Brown working with Bruce Donald in the Cornell Robotics and Vision Laboratory have implemented a software environment [RD92] that permits mobile robots to be programmed using the Scheme programming language [CR92].

We provide brief explanations for some of the mobot control primitives below.

1. `(translate)` and `(rotate)` are high-level routines for controlling the mobot wheel base.

   They can be used with the following keywords:

   - `'by` indicates a desired position termination, and is a signed integer (in millimeters for translate, degrees for rotate).
   - `'vel` tags the desired velocity, a signed quantity.
   - `'accel` specifies the desired acceleration/deceleration to use (unsigned).
   - `'dir` tags the direction of motion (+1 for forward, counterclockwise; -1 for backward, clockwise).
   - `'torque` specifies the maximum torque to apply.
   - `'until` tags a guard (termination predicate), a procedure of one argument. The guard is called with the status of the motion, which is the motor status as reported by the wheel base. There are useful functions called arrived? and impediment? that can be applied to this status value.

   For example:

```
(translate)                  ; moves with default direction, speed,
                             ; acceleration, and torque limit,
                             ; until an impediment is detected

(translate 'by 100)          ; as above, but detects arrival
                             ; after 100mm


(translate 'until (lambda (s)        ; moves until bumper contact
   (not (zero? (read-bumpers)))))    ; or impediment
```

2. `tolerate-impediments` is used around a motion procedure as in

   ```
   (tolerate-impediments (translate 'by dist))
   ```

   to signal when an impediment is detected.

3. `(host-procedure` *proc*`)` defines a remote procedure call to procedure *proc* on the Sparc [RD92].

4. `(reckon)` is an interface procedure to the dead-reckoning software built in to the mobot wheel base. `(reckon)` returns the current mobot position and heading $((x\ .\ y)\ \theta)$ where x and y are given in millimeters, and θ is given in degrees.

B.2 TOMMY's code

Below is the code that was loaded onto the mobile robot TOMMY.

```
(initialize-wb)                ; initialize wheel base

(define reckon-position car)
(define reckon-heading cadr)
(define reckon-x car)
(define reckon-y cdr)

; initialize dead reckoning odometry
(define (set-reckon! x y theta)
    (set-reckon-position! x y)
    (set-reckon-heading! theta)
    (reckon))


(define (small-angle angle)
    (- (modulo (+ angle 180) 360) 180))


; move to a new configuration (x y theta)
(define (goto config)
    (let* ((new-x (car config))        ; x-component of new config
           (new-y (cadr config))       ; y-component of new config
           (new-theta (caddr config))  ; theta-component of new config
           ; get current position and heading
           (current (reckon))
           (x (reckon-x (reckon-position current)))
           (y (reckon-y (reckon-position current)))
           (theta (reckon-heading current))
           ; compute how much to move and rotate
           (dx (- new-x x))
           (dy (- new-y y))
           (move-theta (fixed-atan2 dy dx)))
      (cond ((and (= dx 0) (= dy 0))  ; rotate only
             (rotate 'by (small-angle (- new-theta theta))))
            (#t
             ; rotate towards new position
             (rotate 'by (small-angle (- move-theta theta)))
             ; move to new position
             (translate 'by (isqrt (+ (* dx dx) (* dy dy))))
             ; rotate to new heading
             (rotate 'by (small-angle (- new-theta move-theta)))))))
```

```
; guarded move to a new position (x y)
(define (goto-position config)
    (let* ((new-x (car config))
           (new-y (cadr config))
           (current (reckon))
           (x (reckon-x (reckon-position current)))
           (y (reckon-y (reckon-position current)))
           (theta (reckon-heading current))
           (dx (- new-x x))
           (dy (- new-y y))
           (move-theta (fixed-atan2 dy dx)))
      (rotate 'by (small-angle (- move-theta theta)))
      (tolerate-impediments
           (translate 'by (isqrt (+ (* dx dx) (* dy dy)))))))


; define calling mechanisms to remote procedures
(define match-model (host-procedure 'match-model))
(define new-config (host-procedure 'new-sensor-config))
(define goal-distance (host-procedure 'goal-distance))
(define goal-center (host-procedure 'goal-center))


(define (loop-and-match-model distance iteration)
    ; if an instance of the model is found in the image,
    ; match-model returns the angle to rotate so that the model
    ; is centered within the image
    (let ((correction-angle (match-model distance iteration)))
      (if (and (zero? (read-bumpers)) (not correction-angle))
          (loop-and-match-model distance (+ 1 iteration))
          correction-angle)))


(define (goto-and-match-model config distance)
  (goto config)
  (write (reckon))
  (let* ((iteration 0)
         (correction-angle (loop-and-match-model distance iteration)))
    (if correction-angle
        (say "I see Lilly.")
        (say "Ouch."))
    correction-angle))
```

```
(define (goto-lily)
  (tolerate-impediments (translate 'by (- (goal-distance) mobot-radius)))
  (write (reckon))
  (say "Hi Lilly.")
  (sleep 550)
  (translate 'by (* 2 mobot-radius))
  (rotate 'by (small-angle (- 0 (reckon-heading (reckon)))))
  (translate 'by 3000))


(define (video-demo)
  (set-reckon! 0 0 0)
  (sleep 2000)                          ; 20 seconds to start taping
  (let ((correction-angle (goto-and-match-model (new-config)
                                                (goal-distance))))
    (cond (correction-angle
           (rotate 'by correction-angle)
           (goto-lily))
          (#t
           #f))))


(define (reckon-to-config r)
  (list (reckon-x (reckon-position r))
        (reckon-y (reckon-position r))
        (reckon-heading r)))


(define (finish-demo)
  (let* ((where-you-are (reckon-to-config (reckon)))
         (correction-angle (goto-and-match-model where-you-are
                                                 (goal-distance))))
    (cond (correction-angle
           (rotate 'by correction-angle)
           (goto-lily))
          (#t
           #f))))
```

Bibliography

[AAG+86] T. Asano, T. Asano, L. Guibas, J. Hershberger, and H. Imai. Visibility of disjoint polygons. *Algorithmica*, 1:49–63, 1986.

[Bal91] D. H. Ballard. Animate vision. *Artificial Intelligence*, 48:57–86, 1991.

[BB82] D. H. Ballard and C. M. Brown. *Computer Vision.* Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[BCD93] R. G. Brown, L. P. Chew, and B. R. Donald. Localization and map-making algorithms for mobile robots. In *Proceedings of the International Association of Science and Technology for Development (IASTED) International Conference on Robotics and Manufacturing, Oxford, England*, pages 185–190, September 1993.

[BD94] A. J. Briggs and B. R. Donald. Automatic sensor configuration for task-directed planning. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation, San Diego, CA*, pages 1345–1350, May 1994.

[BKT89] B. Bhattacharya, D. G. Kirkpatrick, and G. T. Toussaint. Determining sector visibility of a polygon. In *Proceedings of the Fifth Annual ACM Symposium on Computational Geometry, Saarbrücken, Germany*, pages 247–253, June 1989.

[BL83] R. Brooks and T. Lozano-Pérez. A subdivision algorithm in configuration space for findpath with rotation. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence, Karlsruhe, Germany*, August 1983.

[Bri89] A. J. Briggs. An efficient algorithm for one-step planar compliant motion planning with uncertainty. In *Proceedings of the Fifth Annual ACM Symposium on Computational Geometry, Saarbrücken, Germany*, pages 187–196, June 1989.

[Bri92] A. J. Briggs. An efficient algorithm for one-step planar compliant motion planning with uncertainty. *Algorithmica*, 8(3):195–208, 1992.

[Bro85] R. C. Brost. Planning robot grasping motions in the presence of uncertainty. Technical Report CMU–RI–TR–85–12, Computer Science Department and the Robotics Institute, Carnegie-Mellon University, 1985.

[Bro86] R. C. Brost. Automatic grasp planning in the presence of uncertainty. In *Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco*, April 1986.

[Bro95] R. G. Brown. *Algorithms for Mobile Robot Localization and Building Flexible, Robust, Easy to Use Mobile Robots.* Ph.D. dissertation, Department of Computer Science, Cornell University, Ithaca, NY, May 1995.

[Buc87] S. J. Buckley. *Planning and Teaching Compliant Motion Strategies.* Ph.D. dissertation, MIT Artificial Intelligence Laboratory, 1987.

[BY92] A. Blake and A. Yuille, editors. *Active Vision.* The MIT Press, 1992.

[Can86] J. F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):34–43, 1986.

[Can87] J. F. Canny. *The Complexity of Robot Motion Planning.* Ph.D. dissertation, Department of Electrical Engineering and Computer Science, MIT, 1987.

[Can89] J. F. Canny. On computability of fine motion plans. In *Proceedings of the IEEE International Conference on Robotics and Automation, Phoenix, AZ*, pages 177–182, May 1989.

[CD88] J. F. Canny and B. R. Donald. Simplified Voronoi diagrams. *Discrete and Computational Geometry*, 3:219–236, 1988.

[CD90] A. J. Cameron and H. Durrant-Whyte. A Bayesian approach to optimal sensor placement. *International Journal of Robotics Research*, 9(5):70–88, 1990.

[CDRX88] J. F. Canny, B. R. Donald, J. Reif, and P. G. Xavier. On the complexity of kinodynamic planning. In *Proceedings of the IEEE Symposium on Foundations of Computer Science, White Plains, NY*, October 1988.

[CG93] J. F. Canny and K. Y. Goldberg. A “RISC” paradigm for industrial robotics. In *Proceedings of the Sixth International Symposium on Robotics Research, Hidden Valley, PA*, October 1993.

[CH92] A. Castaño and S. Hutchinson. Hybrid vision/position servo control of a robotic manipulator. In *Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France*, pages 1264–1269, May 1992.

[CK88] C. K. Cowan and P. D. Kovesi. Automatic sensor placement from vision task requirements. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(3):407–416, May 1988.

[CR87] J. F. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In *Proceedings of the IEEE Symposium on Foundations of Computer Science, Los Angeles, CA*, pages 49–60, October 1987.

[CR92] W. Clinger and J. Rees. Revised^4 report on the algorithmic language Scheme. Technical Report TR 92–1261, Department of Computer Science, Cornell University, January 1992.

[dBGH$^+$93] M. de Berg, L. Guibas, D. Halperin, M. Overmars, O. Schwarzkopf, M. Sharir, and M. Teillaud. Reaching a goal with directional uncertainty. In *Proceedings of the Fourth Annual International Symposium on Algorithms and Computation, Hong Kong*, December 1993.

[DJ91] B. R. Donald and J. Jennings. Sensor interpretation and task-directed planning using perceptual equivalence classes. In *Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento, CA*, 1991.

[DJ92a] B. R. Donald and J. Jennings. Constructive recognizability for task-directed robot programming. *Journal of Robotics and Autonomous Systems*, 9(1):41–74, 1992.

[DJ92b] B. R. Donald and J. Jennings. Constructive recognizability for task-directed robot programming. In *Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France*, May 1992.

[DJR93a] B. R. Donald, J. Jennings, and D. Rus. Experimental information invariants for cooperating autonomous mobile robots. In *International Joint Conference on Artificial Intelligence (IJCAI) Workshop on Dynamically Interacting Robots, Chambery, France*, August 1993.

[DJR93b] B. R. Donald, J. Jennings, and D. Rus. Information invariants for cooperating autonomous mobile robots. In *Proceedings of the Sixth International Symposium on Robotics Research, Hidden Valley, PA*, October 1993.

[DJR94] B. R. Donald, J. Jennings, and D. Rus. Analyzing teams of cooperating mobile robots. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation, San Diego, CA*, pages 1896–1903, May 1994.

[Don87] B. R. Donald. A search algorithm for motion planning with six degrees of freedom. *Artificial Intelligence*, 31(3), 1987.

[Don88] B. R. Donald. The complexity of planar compliant motion planning under uncertainty. In *Proceedings of the Sixth Annual ACM Symposium on Computational Geometry, Urbana*, pages 309–318, June 1988.

[Don89] B. R. Donald. *Error detection and recovery in Robotics*, volume 336 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1989.

[Don90] B. R. Donald. The complexity of planar compliant motion planning under uncertainty. *Algorithmica*, 5(3):353–382, 1990.

[Don91] B. R. Donald. The EIGHT manual: A system for geometric modelling and three-dimensional graphics on the lisp machine. Technical Report TR 91–1186, Department of Computer Science, Cornell University, January 1991.

[Don93a] B. R. Donald. Information invariants in robotics, parts I and II. In *Proceedings of the 1993 IEEE International Conference on Robotics and Automation, Atlanta, GA*, 1993.

[Don93b] B. R. Donald. On information invariants in robotics. Technical Report TR 93–1341, Department of Computer Science, Cornell University, 1993.

[DX94a] B. R. Donald and P. G. Xavier. Provably good approximation algorithms for optimal kinodynamic plans: Robots with decoupled dynamics bounds. To appear in *Algorithmica*, 1994.

[DX94b] B. R. Donald and P. G. Xavier. Provably good approximation algorithms for optimal kinodynamic planning for cartesian robots and open-chain manipulators. To appear in *Algorithmica*, 1994.

[DXCR93] B. R. Donald, P. G. Xavier, J. F. Canny, and J. Reif. Kinodynamic motion planning. *Journal of the ACM*, 40(5), November 1993.

[Ell92] R. E. Ellis. Planning tactile recognition paths in two and three dimensions. *International Journal of Robotics Research*, 11(2):87–111, April 1992.

[Erd84] M. A. Erdmann. On motion planning with uncertainty. Technical Report MIT–AI–TR 810, MIT Artificial Intelligence Laboratory, 1984.

[Erd86] M. A. Erdmann. Using backprojections for fine motion planning with uncertainty. *International Journal of Robotics Research*, 5(1):19–45, 1986.

[Erd93] M. A. Erdmann. Understanding action and sensing by designing action-based sensors. In *Proceedings of the Sixth International Symposium on Robotics Research, Hidden Valley, PA*, October 1993.

[FHS89] J. Friedman, J. Hershberger, and J. Snoeyink. Compliant motion in a simple polygon. In *Proceedings of the Fifth Annual ACM Symposium on Computational Geometry, Saarbrücken, Germany*, pages 175–186, June 1989.

[FHS90] J. Friedman, J. Hershberger, and J. Snoeyink. Input-sensitive compliant motion in the plane. In *Proceedings of the Second Scandinavian Workshop on Algorithmic Theory, Bergen, Norway*, 1990.

[GM87] S. K. Ghosh and D. M. Mount. An output sensitive algorithm for computing visibility graphs. In *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science, Los Angeles, California*, pages 11–19, October 1987.

[GMR92] L. J. Guibas, R. Motwani, and P. Raghavan. The robot localization problem. In *SODA*, 1992.

[Gra72] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1:132–133, 1972.

[GY83] R. L. Graham and F. F. Yao. Finding the convex hull of a simple polygon. *Journal of Algorithms*, 4(4):324–331, 1983.

[HJ94] D. P. Huttenlocher and E. W. Jaquith. Detecting moving objects with a moving camera by comparing edge contours. Technical Report TR 94–1405, Department of Computer Science, Cornell University, January 1994.

[HKR93] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, 1993.

[HLR94] D. P. Huttenlocher, M. E. Leventon, and W. J. Rucklidge. Visually-guided navigation by comparing two-dimensional edge images. In *Proceedings of the Conference on Computer Vision and Pattern Recognition, Seattle, WA*, pages 842–847, 1994.

[HM91] G. Hager and M. Mintz. Computational methods for task-directed sensor data fusion and sensor planning. *International Journal of Robotics Research*, 10(4):285–313, August 1991.

[HR93] D. P. Huttenlocher and W. J. Rucklidge. A multi-resolution technique for comparing images using the Hausdorff distance. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 705–706, New York, NY, 1993.

[Hut91] S. Hutchinson. Exploiting visual constraints in robot motion planning. In *Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento*, pages 1722–1727, April 1991.

[Jar73] R. A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2:18–21, 1973.

[JDC89] J. Jennings, B. R. Donald, and D. Campbell. Towards experimental verification of an automated compliant motion planner based on a geometric theory of error detection and recovery. In *Proceedings of the IEEE International Conference on Robotics and Automation, Phoenix, AZ*, pages 632–637, May 1989.

[JE94] Y.-B. Jia and M. Erdmann. The complexity of sensing by point sampling. In *Workshop on the Algorithmic Foundations of Robotics, San Francisco*, February 1994.

[Jen95] J. Jennings. *Distributed Manipulation With Mobile Robots.* Ph.D. dissertation, Department of Computer Science, Cornell University, Ithaca, NY, May 1995.

[JR93] J. Jennings and D. Rus. Active model acquisition for near-sensorless manipulation with mobile robots. In *Proceedings of the International Association of Science and Technology for Development (IASTED) International Conference on Robotics and Manufacturing, Oxford, England*, September 1993.

[KB93] E. Krotkov and R. Bajcsy. Active vision for reliable ranging: Cooperating focus, stereo, and vergence. *International Journal of Computer Vision*, 11(2):187–203, October 1993.

[KDL94] K. N. Kutulakos, C. R. Dyer, and V. J. Lumelsky. Provable strategies for vision-guided exploration in three dimensions. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation, San Diego, CA*, pages 1365–1372, May 1994.

[KK94] K. Kemmotsu and T. Kanade. Sensor placement design for object pose determination with three light-stripe range finders. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation, San Diego, CA*, pages 1357–1364, May 1994.

[Lat88] J.-C. Latombe. Motion planning with uncertainty: The preimage backchaining approach. Technical Report STAN-CS-88-1196, Department of Computer Science, Stanford University, March 1988.

[Lat91] J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[LD81] D. T. Lee and R. L. Drysdale. Generalization of Voronoi diagrams in the plane. *SIAM J. Comput.*, 10:73–87, 1981.

[LGG92] S. Lacroix, P. Grandjean, and M. Ghallab. Perception planning for a multi-sensory interpretation machine. In *Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France*, pages 1818–1824, May 1992.

[LMT84] 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.

[Loz83] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Trans. on Computers (C–32)*, pages 108–120, 1983.

[LP84] D. T. Lee and F. P. Preparata. Euclidean shortest paths in the presence of rectilinear boundaries. *Networks*, 14:393–410, 1984.

[LS87] D. Leven and M. Sharir. Planning a purely translational motion of a convex object in two-dimensional space using generalized Voronoi diagrams. *Discrete Comput. Geom.*, 2:9–31, 1987.

[LW79] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, October 1979.

[Mar82] D. Marr. *Vision.* W. H. Freeman and Co., San Francisco, 1982.

[Mas81] M. T. Mason. Compliance and force control for computer controlled manipulators. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11(6):418–432, 1981.

[MB92] J. Maver and R. Bajcsy. Occlusions and the next view planning. In *Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France*, pages 1806–1811, May 1992.

[MMP+91] J. Matoušek, N. Miller, J. Pach, M. Sharir, S. Sifrony, and E. Welzl. Fat triangles determine linearly many holes. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, pages 49–58, 1991.

[Nat86] B. K. Natarajan. *On Moving and Orienting Objects.* Ph.D. dissertation, Department of Computer Science, Cornell University, Ithaca, NY, 1986.

[NK94] B. Nelson and P. K. Khosla. Integrating sensor placement and visual tracking strategies. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation, San Diego, CA*, pages 1351–1356, May 1994.

[O'R87] J. O'Rourke. *Art Gallery Theorems and Algorithms.* Oxford University Press, New York, 1987.

[OSY86] C. O'Dúnlaing, M. Sharir, and C. Yap. Generalized Voronoi diagrams for a ladder: I. Topological considerations. *Communications in Pure and Applied Mathematics*, 39:423–483, 1986.

[OSY87] C. O'Dúnlaing, M. Sharir, and C. Yap. Generalized Voronoi diagrams for a ladder: II. Efficient construction of the diagram. *Algorithmica*, 2:29–57, 1987.

[OY85] C. O'Dúnlaing and C. Yap. A "retraction" method for planning the motion of a disc. *Journal of Algorithms*, 6:104–111, 1985.

[Ped63] D. Pedoe. *An Introduction to Projective Geometry.* The Macmillan Company, 1963.

[PSS86] R. Pollack, M. Sharir, and S. Sifrony. Separating two simple polygons by a sequence of translations. Technical Report 215, Department of Computer Science, New York University, Courant Institute of Mathematical Sciences, April 1986.

[RB92] R. Rimey and C. Brown. Task-oriented vision with multiple Bayes nets. In A. Blake and A. Yuille, editors, *Active Vision*, chapter 14, pages 217–236. The MIT Press, 1992.

[RD92] J. Rees and B. R. Donald. Program mobile robots in scheme. In *Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France*, May 1992.

[Rei79] J. H. Reif. The complexity of the Movers problem and generalizations. In *Proceedings of the IEEE Symposium on Foundations of Computer Science, San Juan, PR*, pages 421–427, 1979.

[Ruc95] W. J. Rucklidge. *Efficient Computation of the Minimum Hausdorff Distance for Visual Recognition*. Ph.D. dissertation, Department of Computer Science, Cornell University, Ithaca, NY, January 1995.

[SAS84] M. Sharir and E. Ariel-Sheffi. On the piano movers' problem: IV. Various decomposable two-dimensional motion planning problems. *Communications in Pure and Applied Mathematics*, 37:479–493, 1984.

[SH94] R. Sharma and S. Hutchinson. On the observability of robot motion under active camera control. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation, San Diego, CA*, pages 162–167, May 1994.

[SL91] S. Shekhar and J. C. Latombe. On goal recognizability in motion planning with uncertainty. In *Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento*, pages 1728–1733, April 1991.

[SO86] S. Suri and J. O'Rourke. Worst-case optimal algorithms for constructing visibility polygons with holes. In *Proceedings of the Second Annual ACM Symposium on Computational Geometry*, pages 14–23, 1986.

[SS83a] J. T. Schwartz and M. Sharir. On the piano movers' problem: I. The case of a rigid polygonal body moving amidst polygonal barriers. *Communications in Pure and Applied Mathematics*, 36:345–398, 1983.

[SS83b] J. T. Schwartz and M. Sharir. On the piano movers' problem: II. General techniques for calculating topological properties of real algebraic manifolds. *Advances Appl. Math.*, 4:298–351, 1983.

[SS83c] J. T. Schwartz and M. Sharir. On the piano movers' problem: III. Coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers. *International Journal of Robotics Research*, 2(3):46–75, 1983.

[SS84] J. T. Schwartz and M. Sharir. On the piano movers' problem: V. The case of a rod moving in $3D$ space amidst polyhedral obstacles. *Communications in Pure and Applied Mathematics*, 37:815–848, 1984.

[SS86] M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. *SIAM Journal of Computing*, 15:193–215, 1986.

[SS93] M. J. Swain and M. A. Stricker. Promising directions in active vision. *International Journal of Computer Vision*, 11(2):109–126, October 1993.

[Tel92] S. J. Teller. Computing the antipenumbra of an area light source. *Computer Graphics (Proceedings SIGGRAPH '92)*, 26(2):139–148, July 1992.

[TT92] K. Tarabanis and R. Y. Tsai. Computing occlusion-free viewpoints. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 802–807, 1992.

[Wel85] E. Welzl. Constructing the visibility graph for n line segments in $O(n^2)$ time. *Information Processing Letters*, 20:167–171, 1985.

[Whi77] D. Whitney. Force feedback control of manipulator fine motions. *Journal of Dynamic Systems, Measurement, and Control*, 99(2):91–97, June 1977.

[Xav92] P. G. Xavier. *Provably-Good Approximation Algorithms for Optimal Kinodynamic Robot Motion Plans*. Ph.D. dissertation, Department of Computer Science, Cornell University, Ithaca, NY, May 1992.

[Zha92] H. Zhang. Optimal sensor placement. In *Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France*, pages 1825–1830, May 1992.