June 1993

Report No. STAN-CS-93-1477

Thesis

PB96-149257

Contingency-Tolerant Robot Motion Planning and Control

by

Wonyun Choi

Department of Computer Science

Stanford University

Stanford, California 94305

CONTINGENCY-TOLERANT ROBOT MOTION PLANNING AND CONTROL

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

By
Wonyun Choi
June 1993

Abstract

This dissertation describes a new approach and enabling techniques to build the navigation system of a mobile robot operating in a partially known environment. The main layout of this environment is known in advance, but the locations and shapes of some smaller objects may not be known. Environments of this type include shop-floors, clean rooms, offices, etc. The problem is to automatically determine how the robot should move from one position to another without colliding with any of the objects in the environment.

In order to be both efficient and robust, the navigation system should interweave a *planning component* and a *reaction component*. The planning component should take advantage of available prior knowledge to produce globally efficient plans. However, it should be aware that knowledge may be incomplete and generate lesser-committed plans that leave some freedom of choice at execution time to deal with contingencies. The reaction component should organize the robot's behavior according to both the ongoing plan and the sensory inputs.

The main idea underlying the approach proposed in this dissertation is to let the reaction component share some global knowledge of the robot's environment with the planning component. Based on this idea, we have developed a new type of navigation system where the planning component generates a lesser-committed motion plan, called a *channel*, represented by a sequence of parallelepipedic cells, and the reaction component uses artificial potential fields to pull the robot toward its goal within the channel, while repelling it away from unexpected obstacles.

Various arrangements of unexpected obstacles may be encountered during navigation. The treatment of these arrangements by the reaction component is organized in

three layers: channel navigation, local replanning, and global replanning. The most common and simpler cases are processed by the first layer using less complex techniques (tracking an artificial potential field), while less frequent but more complex cases are handled by the other layers using more complex techniques (local/global replanning).

This navigation system applies to mobile robots with holonomic, as well as nonholonomic constraints. Three versions of the system have been implemented. They have been experimented with simulated robots, a Robotworld system, and the GOFER mobile robot, respectively. Experimental results are discussed in this report.

Acknowledgements

First, I wish to thank my advisor, Professor Jean-Claude Latombe for his insightful suggestions, encouragement and generous support throughout this research. This research could never have been accomplished without his energetic guidance. I also appreciate his endless efforts to create an ideal research atmosphere at the Computer Science Robotics Laboratory. I thank other members of my reading committee, Professors Mark Cutkosky, Oussama Khatib, and Larry Leifer, for serving on the committee and for their thorough review.

I am indebted to my principal advisor in Mechanical Engineering Department, Professor Leifer, for his support during the early years of my stay at Stanford. I am further indebted to Professor Khatib for his pioneering work on Artificial Potential Field method and his inspiring suggestions.

I am especially grateful to David Zhu, Mark Yim, Bryant Marks, Jim Slater, Yan Martini, and Tsai-Yen Li, with whom I worked closely. David Zhu provided codes for channel generation, and made constructive criticisms of this dissertation. Mark Yim, Bryant Marks, and Jim Slater spent countless days and nights for designing and building the mobile robot GOFER. Yan Martini and Tsai-Yen Li built the experimental setup on Robotworld, and provided codes for controlling the manipulation robots in Robotworld.

This research has been partially funded by DARPA contract DAAA21-89-C0002 (Army). It also benefited from a grant of Digital Equipment Corporation and a grant of CIFE (Center for Integrated Facility Engineering).

Finally, I would like to thank all of my family members for their support, devotion, and patience. It is to my parent that I dedicate this dissertation.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

During the past twenty years many advances have been made in mobile robotics. The ultimate goal of such robots is to operate autonomously and achieve high-level goals describing what should be done, rather than how to do it. Various application tasks have been considered, for example, delivering mail, carrying luggage, guiding people, surveilling offices, acquiring information. To accomplish such tasks successfully, the very basic thing that a robot must be able to do is to navigate reliably from one location to another. Yet, reliable navigation remains a standing problem. This is due to the fact that real world is full of uncertainties and contingencies that are difficult to model meaningfully, e.g., motion control is imperfect, sensors are inaccurate and/or may not work properly, knowledge of the world is incomplete. Such imperfections yield events that the robot may not handle appropriately or even recognize. It is definitely not reasonable to expect that a robot can face all sorts of unexpected events (humans certainly can't). Nevertheless, for virtually any environment, it should be possible to define a finite set of event types covering all events that may reasonably occur. In such an environment, a reliable robot is one that behaves appropriately when these events occur. Notice that these events are "expected" in the sense that it is known in advance that they may occur. But they are "unexpected" in the sense that one cannot predict when and/or how they will occur.

In this dissertation, we consider the case of an office-assistant mobile robot.[1]

[1]Office robots are only used here as a source of inspiration and an illustrative example. Most of

Clearly, such a robot cannot have or maintain complete knowledge of its workspace. Hence, a major type of events is caused by "unexpected obstacles," that is, obstacles which are not part of the robot's workspace model. In this dissertation, we focus on handling this type of events. We address the following question: EXPECTING THE EXISTENCE OF UNEXPECTED OBSTACLES, HOW CAN THE NAVIGATION SYSTEM OF A MOBILE ROBOT EFFICIENTLY DEAL WITH THESE OBSTACLES?

1.1 Motivation

The problem of planning and executing motions in an incompletely known environment is an important one in robotics. It occurs in various applications such as transportation tasks in shop-floors, office environments, clean rooms and construction sites. In all these workspaces, substantial knowledge exists in advance; but it is unrealistic or too constraining to assume that this knowledge is accurate and complete. In fact, there exist very few applications where prior knowledge is either complete, or totally nonexistent. Incomplete knowledge requires the navigation system to combine planning and reactive capabilities. This combination is under active study in the artificial intelligence community [Fir87, GL87, Kae86], but often at a high level of task performance. So far, it has made few inroads at the robot motion planning and control level.

Most existing robot navigation systems fall into either one of two classes:

- Systems of the first class plan a navigation path assuming complete knowledge and then execute the path. If a significant difference with the planning model is perceived during execution, they stop the motion and replan a path using the additional knowledge acquired by the sensors. Most path planners (e.g., [BL91, BLP85, LPW79, SS83]) can be used to build navigation systems of this class.

  The drawbacks of these systems are:

our work can be extended to other indoor robots and, to a lesser extent, outdoor robots.

- They produce over-constrained plans that may cause frequent replanning operations.

- They require dedicating significant processing power to check the difference between the planning model and the real world.

Moreover, uncertainties in sensing and control make it difficult to decide whether a difference is significant enough to terminate a path and replan a new one.

- Systems of the second class lie at the other extreme. They assume no prior knowledge. They pilot the robot using only local information provided by the sensors and/or the workspace model. Examples of such systems include potential-field-based controllers [Kha86] and boundary-following controllers [LS86].

  Because they use only local information, these systems either lack completeness (i.e., they may fail even if the goal is reachable), or may often be very inefficient. Potential-field-based systems, which follow the steepest descent of a potential field, may get trapped into obstacle concavities. Boundary-following systems, which track the contours of the obstacles, may explore a large subset of the workspace at every motion.

  These systems lack a planning component to take advantage of the available prior knowledge and guide the global behavior of the robot.

Neither of these two approaches alone allows a mobile robot to efficiently navigate in an incompletely known environment. In this thesis we present a new approach to robot navigation dealing with unexpected obstacles; we describe an implemented system based on this approach; and we show experimental results obtained with this system. Our navigation approach yields a hierarchical architecture interweaving planning and reaction components. We will outline our approach and architecture into more detail later in this introductory chapter. First, however, we present more precisely the navigation problem considered in this dissertation.

1.2 Navigation Problem

We address the problem of mobile robot navigation in an incompletely known workspace. This environment contains two types of obstacles: those which are part of the robot's model (i.e., the robot knows the geometry and position of these obstacles), and those which aren't. The former are called *known obstacles*. The latter are called *unexpected obstacles*, though their existence is expected. Known obstacles are known with sufficient accuracy to allow gross navigation. Over planning/execution cycles unknown obstacles may become known obstacles through some learning process, but such a learning process is not studied here.

Typically, known obstacles are large, fixed, or hardly movable. In an office environment, they include the building walls as well as large pieces of furniture, e.g., desks, copying machines. Unexpected obstacles are usually small and easily movable, e.g., chairs, trash bins. It is not completely ruled out, however, that some unexpected obstacles are large.

All obstacles, both known and unexpected, are stationary. Later in this report, we will discuss how one may try to relax this assumption.

The mobile robot is equipped with sensors (e.g., proximity range sensors) that can detect obstacles located within some limited distance. Hence, these sensors only provide *local* information.

The problem for the robot is to reliably and efficiently navigate from one given location to another. As much as possible, the robot should use its partial knowledge of the workspace to avoid wandering around. Yet, it should be reactive to the unexpected obstacles and not collide with them. As much as possible, it must avoid stopping (even for short amounts of time) when unexpected obstacles bar its route.

Although sensory data are not perfect, we assume that they allow the robot to locate itself at any time with enough precision to perform the navigation and make the right decisions. Without this assumption, the robot system would have to plan motion strategies guaranteeing that enough sensory information will be collected at execution

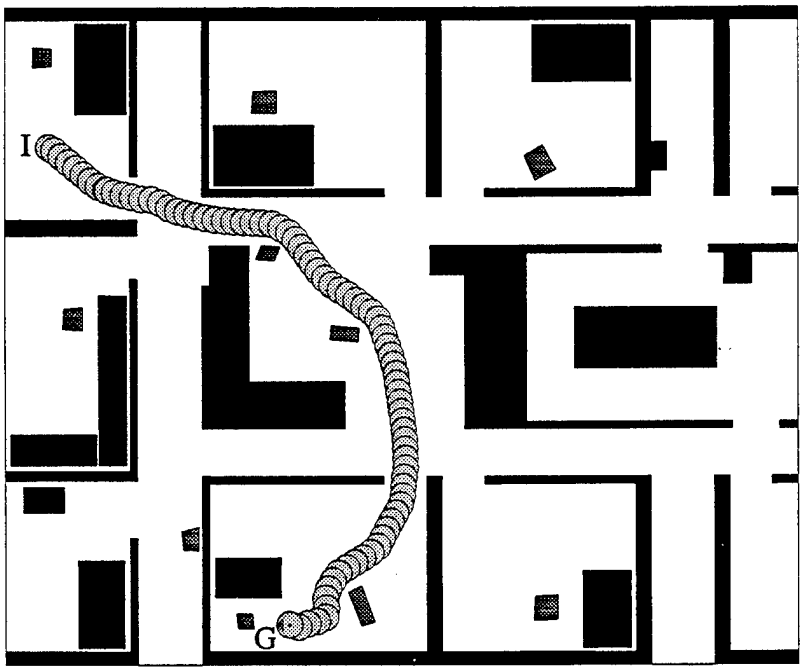


Figure 1.1: Mobile robot navigation with incomplete knowledge

time to determine the robot's location with sufficient precision. The generation of such strategies in the absence of unexpected obstacles is explored in [TL92] and [LL92]. It is not considered here. In other words, collision avoidance and motion efficiency are our only concerns in this thesis.

Figure 1.1 illustrates the above problem with an example. Known obstacles are shown black, unexpected ones are displayed in grey. The figure shows an acceptable navigation path followed by the robot between two given locations, I and G.

1.3 Approach

A navigation system that operates with incomplete knowledge should interweave a *planning component* and a *reaction component.* The planning component should take advantage of available knowledge to produce globally efficient plans to achieve specified goals. However, it should be aware that knowledge is incomplete in order to generate *lesser-committed motion plans*, i.e., plans that leave some freedom of choice

at execution time. The reaction component should organize the robot's behavior according to both the ongoing plan and the sensory inputs.

The main idea underlying our approach is to let the reaction component share some global knowledge of the robot's environment with the planning component. This knowledge is contained in the lesser-committed plan generated by the planning component. In more traditional navigation approaches, the planning and reaction components are organized sequentially in such a way that the global prior knowledge is exclusively accessed by the planning component for creating motion plans. The reaction component's only source of information consists of the local information provided by the sensors. The result is that motion changes caused by this information are decided independently of any desired global behavior, and therefore may often be inconsistent with it. In our approach, the reaction component has access to some global knowledge contained in the lesser-committed plan. This knowledge has been "compiled" by the planning component in order to make it usable in real time by the reaction component. The reaction component determines the appropriate robot's reaction to unexpected obstacles detected by sensors *in the context* of the compiled global knowledge. Moreover, this approach frees the planning component from having to arbitrarily select a specific motion path that unexpected obstacles could rapidly make obsolete.

But:

- What is a lesser-committed motion plan?
- How can a reaction system make use of it?

Most of this report is aimed at giving precise answers to those two questions. Let us just say here that a lesser-committed plan is a continuous set of contiguous paths that are free of collisions with the known obstacles. We call such a set a *channel*.[2] There is no guarantee that a channel does not intersect unexpected obstacles. But, if

[2]Actually, a channel is the main form of a lesser-committed plan used in our navigation system, but it is not the only one.

these obstacles are small (as should often be the case), it is likely to contain some paths that are free of collision with any of them. In other words, the channel leaves some freedom of choice at execution time, while providing global information about what is likely to be a suitable direction of motion. Reaction is organized in a channel by using artificial potential fields that pull the robot toward its goal *within* the channel, while repelling it away from the unexpected obstacles. At every instant, the robot is commanded to move along the negated gradient of the total potential field, as if it was a particle under the influence of this field [Kha86]. Figure 1.2 is a simplified diagram of the information flow between the planning and the reaction components.

However, various situations may be encountered. For instance, though most unknown obstacles are likely to be small and scattered, a number of small obstacles may have been gathered together. Big unexpected obstacles are not impossible, either. This may seriously complicate reactive motion in a channel, for example by creating local minima of the artificial potential field. Without global knowledge of the unexpected obstacles, such a contingency cannot be fully eliminated. Thus, *local replanning* may be needed to generate refined lesser-committed plans *within the current channel.*[3] It cannot be excluded either that unexpected obstacles completely obstruct a channel (for example, such an obstacle may be a big piece of furniture temporarily pushed across a corridor). Then it may become necessary to *replan a new channel.*

Obviously, all situations are not equally likely. In a typical office environment, unexpected obstacles are usually small and sparsely scattered around. Less often, but still quite frequently, some are gathered together (e.g., chairs). More rarely, big unexpected obstacles obstruct passageways. Although the navigation system should handle all these situations, it should not waste time hypothesizing that the worst situation occurs at every instant. Thus:

- How should computation be organized so that the most common situations are handled as quickly as possible?

[3]We will see that, in our navigation system, these refined plans are not constructed as channels, though they could be.

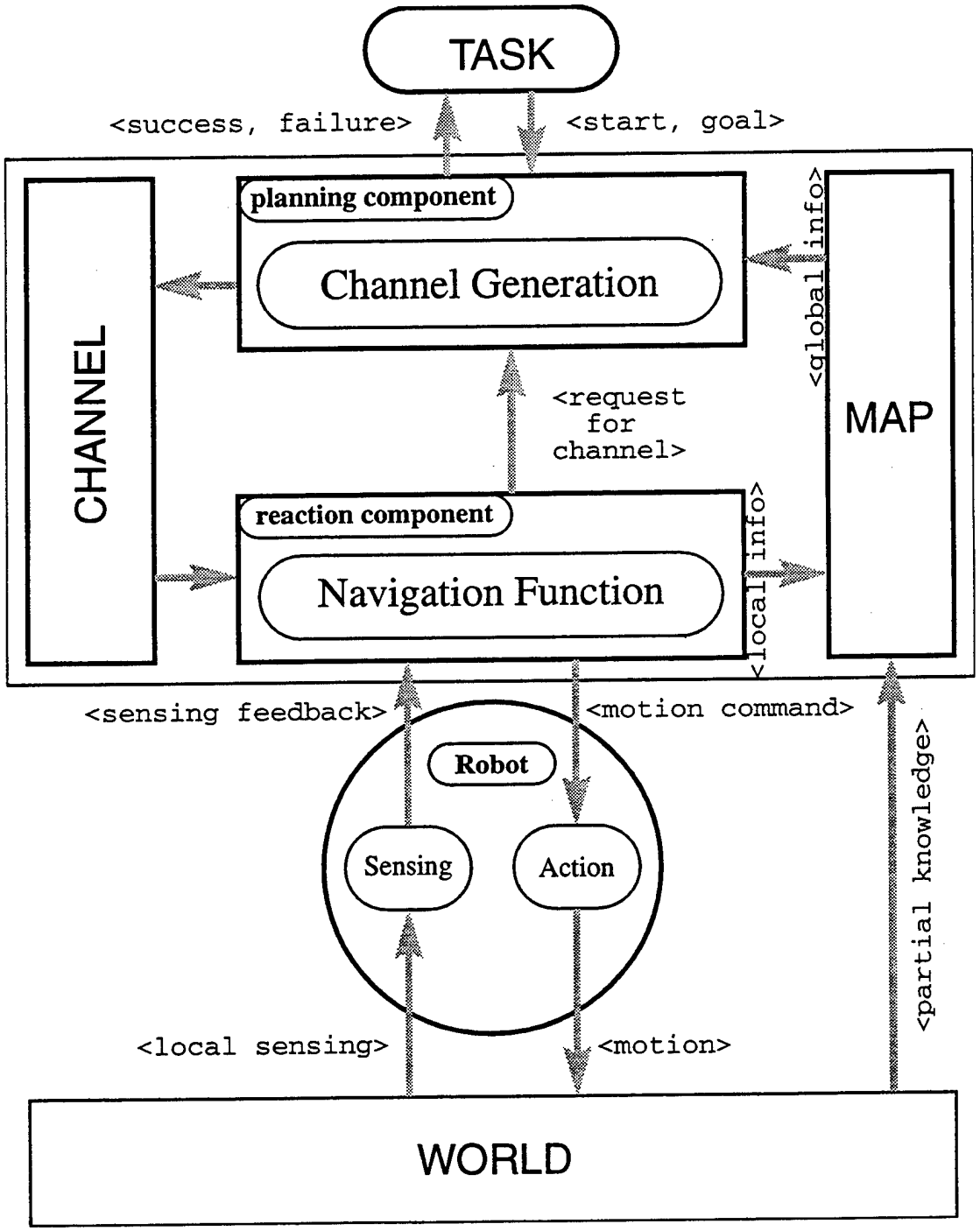


Figure 1.2: Information flow between the planning and reaction components

Another large part of this thesis is devoted to answering this question (including the problem of recognizing situations). Our approach here is to organize both planning and reaction into a hierarchy of modules interweaving their execution. The highest layer treats the most common situations efficiently. Alone, it could also recover from any contingency whose occurrence is expected, but not always efficiently. The lower layers provide more efficient treatment of the less common contingencies. This architecture provides opportunities for incremental performance improvements.

A navigation system based on the above approach can be made complete under the assumptions that knowledge of known obstacles is accurate, all obstacles are stationary and sensing is perfect. Completeness means that, if it is possible to attain the goal, the robot will ultimately find a path; otherwise it will report failure. Of course, completeness also depends on the particular techniques embedded in this architecture.

1.4 Technical Issues

The above approach raises a variety of technical issues. This section briefly analyzes these issues. The following chapters will address them in detail.

1.4.1 Channel Definition and Generation

In an partially known environment, a path defining the continuous sequence of positions and orientations of the robot is an overly constrained motion plan to go from one location to another. Indeed, there usually exist many different paths connecting the two locations which avoid collision with the known obstacles. Some of these paths may lead the robot toward unexpected obstacles. Others may not. But, at planning time, there is no way to know which paths are good and which ones are not. Rather than generating a single path, we propose that the planner constructs a channel, i.e., a set of contiguous paths connecting the initial and goal locations without collision with the known obstacles.

However, the definition of a channel as a set of contiguous paths is not sufficient

to characterize a satisfactory channel. Indeed, this loose definition and the least-commitment arguments aimed at giving maximal freedom of choice to the reaction component would simply lead to considering the whole free space (the set of robot configurations not colliding with any known obstacle) as a channel. But, in general, this set has a complex geometry and topology, and it would not be easy for the reaction component to exploit it in real time. Arbitrarily simplifying the geometry of a channel is not the answer either.

We have identified three criteria to guide the choice of the general geometry of a channel:

1. This geometry should yield efficient channel generation (i.e., global planning).
2. It should yield large channels leaving significant freedom to the reaction component.[4]
3. It should allow fast computation of motion commands by the reaction component.

These criteria led us to define a channel as a *sequence of adjacent parallelepipedic cells* in the robot's configuration space.[5] As we will see, artificial potential fields can easily be defined and computed in a channel having such geometry. Moreover, in a low-dimensional configuration space (which is the case for a mobile robot), such a channel can be efficiently generated by an *approximate cell decomposition* planning method [Lat91]. Actually, when it is used to plan a path, such a method first produces a channel and then extracts a path from this channel. In contrast, other planning approaches, such as roadmap and potential-field planning approaches [Lat91], directly generate a single path. Using them here would require us to add a postprocessing step transforming the generated path into a channel. An exact cell decomposition method

[4]Because we do not require a channel to leave maximal freedom to the reaction component, we call it a "lesser-committed plan", rather than a least-committed plan.

[5]We recall that the *configuration* of a robot is a list parameters representing the position and orientation of the robot. In the case of a non-circular robot, it consists of three parameters (two for the robot's position, one for its orientation). Then the configuration space is three-dimensional. See Chapter 2 for more detail.

would in general produce larger channels than approximate cell decomposition. But the more complex geometry of these channels would conflict with the third criterion above. On the other hand, the loss of free space resulting from the approximate decomposition is not dramatic for gross navigation and is acceptable as long as the robot is not required to make contact (e.g., docking) with obstacles. Furthermore, unlike exact cell decomposition, approximate cell decomposition allows us to easily take some uncertainty in robot control into account, e.g., by imposing a minimal-size requirement on both the cells and the intersection of any two consecutive cells. This is a major practical advantage in order to run experiments with real robots.

1.4.2 Design of Potential Functions

A channel defines a continuous set of paths. The actual path followed by the robot is selected by the reaction component. This component computes an artificial potential field over the channel and makes the robot track the negated gradient of this potential. The artificial potential must depend on the specific shape of the planned channel in order to produce motion commands that are consistent with the global plan represented by this channel. It must also depend on the sensed unexpected obstacles to provide needed reaction to these obstacles. The artificial potential and its gradient are computed on-line while the robot is moving. At every instant, the gradient determines the instantaneous evolution of the specific path followed by the robot.

The potential field must achieve three functions:

1. Drive the robot toward the goal.

2. Keep the robot within the channel.

3. Avoid collision with unexpected obstacles.

Furthermore, as much as possible, the potential field should have no local minima where the robot could get stuck.

These considerations led us to define the potential function as a weighed sum of three elementary potentials:

- An *intermediate-goal potential* attracts the robot through intermediate goal configurations chosen along the channel. It is intended to make the robot progress toward the goal.

- A *channel-wall potential* repels the robot away from the boundary of the channel. To prevent the robot from moving out of the channel, the magnitude of this potential increases to infinity as the distance to the boundary tends to zero. To facilitate navigation in the channel, this potential vanishes beyond some distance of the boundary. This distance is selected in each cell of the channel according to the size of this cell.

- An *unexpected-obstacle potential* repels the robot away from the sensed unexpected obstacles. Its magnitude increases toward infinity when the sensed distance to an unexpected obstacles tends toward zero. It vanishes when the distance to unexpected obstacles is large enough.

The intermediate-goal potential and the channel-wall potential are designed such that their combination, the *channel potential*, is free of local minima. Thus, when there are no unexpected obstacles lying in a channel, the robot navigates through the channel without a hitch. In the presence of non-expected obstacles, the addition of the unexpected-obstacle potential does no longer guarantee that this is true.

1.4.3 Local Replanning

Sensing unexpected obstacles causes a repulsive component (the unexpected-obstacle potential) to be added to the potential function. The purpose of adding this repulsive component is to make the robot navigate around these obstacles. However, as mentioned above, it may result in a total potential function that has local minima, and there seems to be no ways to prevent this from happening. Fortunately, as long as the unexpected obstacles are small and sparsely scattered, the attraction wells of these

minima (if any) are small, so that it is rather unlikely that the robot gets trapped into any one of them.

Nevertheless, it would not be reasonable to completely disregard local minima. First, even if the attraction domain of a local minimum is small, the robot may nevertheless enter it. Second, several small unexpected obstacles close to each other (e.g., two chairs) or a large one may form a trap and produce a local minimum with a larger attraction basin. One way to deal with such a minimum is to replan a completely new channel connecting the robot's current location to the goal using the detected unexpected obstacles as additional world knowledge. This could be done using the previous planner, but would often be too time-consuming. An alternative is to take advantage of the existing channel and to *locally refine* it into a smaller one not containing the detected unexpected obstacles, if one such refined channel exists. This approach, however, has two main drawbacks:

- It heavily relies on sensing being accurate. Noise in sensing and thereby imperfect models may result in an incorrect refined channel. This becomes especially critical in cluttered areas of the workspace.

- Channel refinement tends to yield narrow channels. In such channels, potential-field-based navigation is difficult, because the channel-wall repulsive potential is permanently active causing the robot to oscillate about its main route.

Instead, we propose to use a fast potential-field planning method (namely, the method described in [BL91]) to generate a *local path* in the channel among the unexpected obstacles. This path is constructed in the cell (exceptionally, in two consecutive cells) of the channel where the robot is currently located and connects the current location of the robot to the next cell (hence, the term "local path"). Because planning is restricted to a small area, it can be made extremely fast. The generated local path, however, may be too committed a plan. We soften this commitment by transforming the path into a valley-shaped potential that is added to the previous potentials. Intuitively, this potential is shaped like the valley of a river that would follow the path. Hence, it does not constrain the robot to exactly follow the generated local path.

Above, we did not mention one important issue: How are local minima detected? Typically, a robot never exactly attains a local minimum. Because of discretized control, it tends to loop around it. Local minima can thus be detected when the gradient of the potential field abruptly changes direction. This may be too conservative a test, but local replanning is so fast that it is preferable to detect some wrong local minima than to let the robot waste time at any actual minimum.

1.4.4 Global Replanning

To get a reliable navigation system, big unexpected obstacles that completely obstruct a channel cannot be totally excluded. Such obstacles result in a local minima that the robot cannot miss, nor escape by staying in the channel. However, when such a local minimum is attained, only a subset of the corresponding obstacle has usually been sensed. Hence, there is no way for the robot to immediately recognize that the channel is fully obstructed.

The robot escapes such a minimum as any other local minimum. It then naturally falls into a second minimum that it tries to escape again, and so on, until enough sensory information has been accumulated that local replanning fails to find a path. Then, the navigation system calls back the original planner to construct a new channel connecting the current robot location to the goal one with the detected unexpected obstacles added to the world model. If no new channel can be found, the navigation system reports global failure.

The number of local minima successively encountered before a new channel is generated depends on the size of the channel cell and the obstructing obstacle, and on the robot's sensors. The more local the sensors, the longer it takes to recognize obstruction. In environments where channel obstruction is frequent, it could be more efficient to skip local replanning and, instead, directly call back the channel planner whenever a local minimum is encountered.

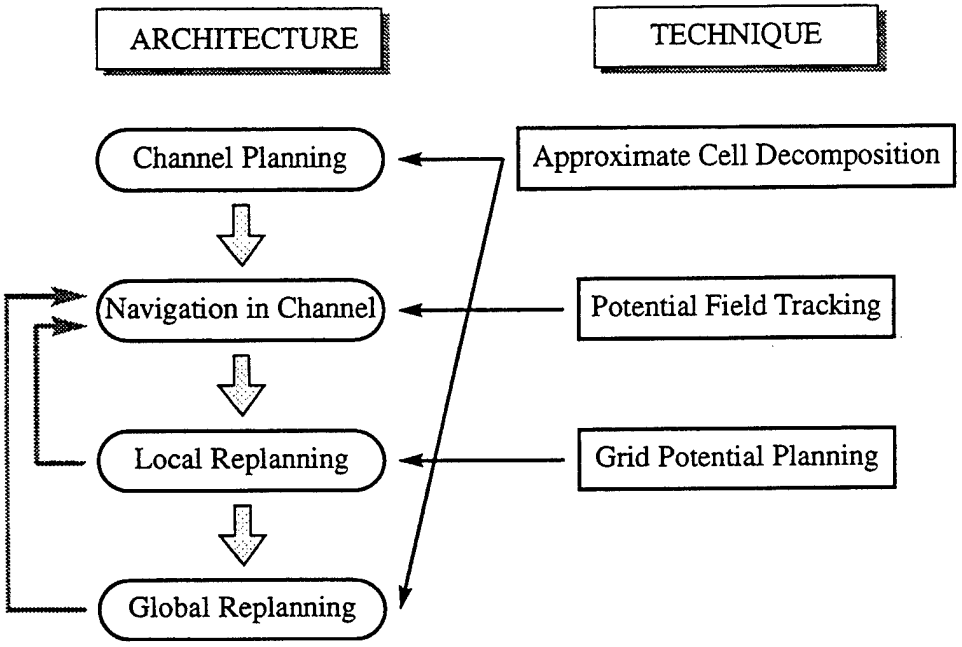


Figure 1.3: Architecture and techniques of the navigation system

1.4.5 Summary

Figure 1.3 shows the architecture of the proposed navigation system. In addition to the main planner, it has three layers, *channel navigation*, *local replanning*, and *global replanning*. Each layer handles the navigation problem at a different complexity. The most common and simpler cases (navigation in channel) are processed by the less complex technique (tracking an artificial potential field), while less frequent but more complex cases (escaping local minima) are handled by more complex techniques (local/global replanning).

1.5 Example

Figure 1.4 shows several paths of a robot generated by our navigation system.

(a) shows a typical workspace with known obstacles (shown black), and the initial and goal configurations of the robot (I and G).

(b) shows a 3-dimensional channel, where the vertical dimension represents the

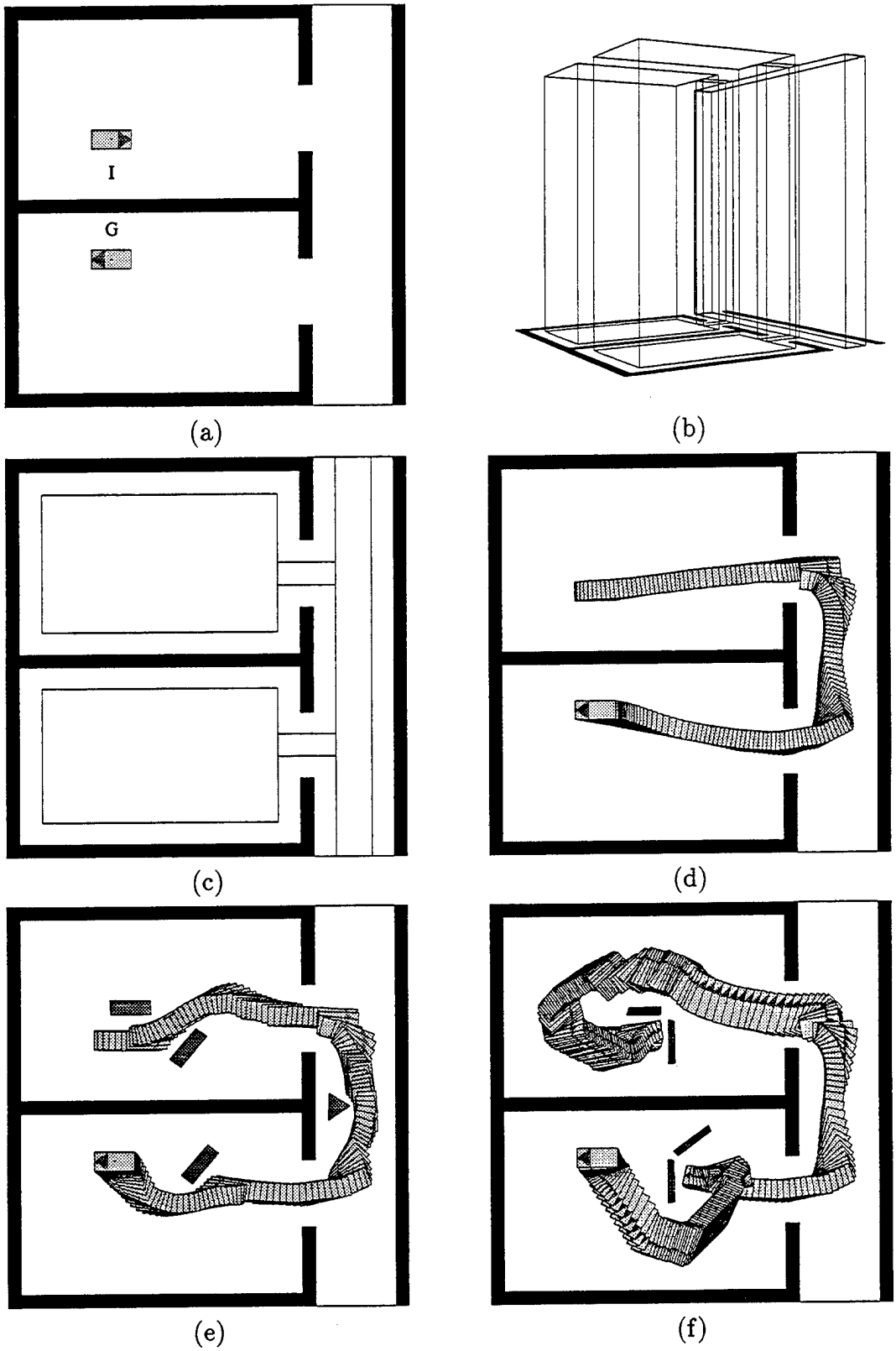


Figure 1.4: Channel and navigation paths

robot's orientation. In this simple channel, every parallelepipedic cell extends over the interval $[0, 2\pi]$, allowing the robot to take any orientation.

(c) shows the projection of the channel in the 2-dimensional workspace. When the center of the robot lies in this channel, the robot collides with none of the known obstacles, whatever its orientation.

(d) shows the path of the robot when it navigates in the channel with no unexpected obstacles.

(e) shows the path of the robot when it navigates in the channel with unexpected obstacles (shown grey) not causing any local minimum.

(f) shows the path of the robot when it navigates in the channel with unexpected obstacles causing local minima, which are escaped by local replanning.

1.6 Implementation

Our navigation system has been tested and verified on a computer-simulated robot and two real robot systems, Robotworld and GOFER.

A computer simulation system was developed on NeXT computer using C and LISP programming languages. It has a graphical user interface, and was used mainly for the algorithm development/verification and for studying the sensitivity of the system to various parameters and setting the values of these parameters.

Robotworld was used to test the navigation system for a robot that can translate and rotate simultaneously, without constraint, on its planar workspace. Robotworld consists of several manipulation robots. These robots, supported by electric magnets and air-cushion, translate under a planar ceiling. Combined with its gripper, each manipulation robot can provide three-degree-of-freedom motions (translation and rotation) in the plane for an object it manipulates.[6] One of the robots holds an object with its gripper and moves it in a planar workspace crowded with obstacles. Both

[6]In fact, the gripper can also provide an additional degree of freedom (vertical motion), but we do not use it for navigation experiments.

planning and reaction component of the navigation system runs on a remote host computer, and the motion commands for the robot are sent to the Robotworld's controller via a serial line interface. The description of both known and unknown obstacles is specified to the host computer, but only the information about known obstacles is provided to the planning component for the generation of the channel. The information about unknown obstacles is not directly accessible to the reaction component; but it is provided during navigation through computer-simulated proximity sensors running simultaneously on the host computer.

GOFER was used to demonstrate the navigation system in a real office-like environment. GOFER is a mobile robot developed in the Computer Science Robotics Laboratory of Stanford University. It consists of a three-wheeled two-degrees-of-freedom mobile base, a ring of infra-red proximity sensors, a laser-camera ranging system, touch sensors and on-board computer. Like a car, GOFER is subject to non-holonomic constraints restricting its linear velocity to the direction its wheels are aligned with. Unlike a car, however, GOFER can turn with arbitrary turning-radii (including zero). These kinematic constraints are taken into account by the reaction component (but not by the planning component) to generate of smooth motion commands. The planning component of the navigation system runs on an off-board computer and the reaction component runs on the on-board computer. The description of the planned channel is sent to the GOFER's on-board computer through a radio modem. For the experiments, we used infra-red proximity sensors to detect obstacles.

1.7 Related Work

1.7.1 "Historical" Systems

Research on mobile robots began in the late sixties with the Stanford Research Institute's pioneering work. Two versions of SHAKEY were built in 1968 and 1971, and they were used as a tool for research in planning and learning [Nil69].

In the late seventies, Moravec started his work on mobile robots with the Stanford Cart [Mor77]. The Cart was a minimal computer-controlled mobile platform. It used stereo-vision to locate objects and deduce its own motion. In the early eighties, the Stanford Mobile Robot, also known as MOBI, was used as a testbed for world modeling and navigation with stereo-vision [KTB89]. Since the late eighties, a series of mobile robots called GOFER have been developed for indoor automation [CCL+90]. Their main goal is to navigate in an office environment and perform tasks such as delivering small objects, guiding/following people and surveilling offices.

Other *historical* mobile robot projects were developed at various organizations. The HILARE project started in the late seventies at LAAS (Toulouse) [Gir79]. The project goal was to perform general research in robot perception and planning. Moravec continued his work at Carnegie-Mellon University with a more capable mobile robot, the CMU Rover [Mor83]. Since 1985, the MIT Mobile Robot group has advocated a radically different architecture for autonomous mobile robots [Bro86]. They build a reactive architecture, called "Subsumption architecture", by stacking up layers of primitive goal-achieving behaviors formed from precompiled sensor to actuator transformations. This new approach has been implemented on a group of MIT robots, called MOBOTs, such as *Allen*, *Hebert*, *Tom* and *Jerry* [FB88].

Some mobile robot projects explicitly focused on navigation in outdoor environments. A series of outdoor robots, including NAVLAB, were developed at Carnegie-Mellon University. NAVLAB was built based on a commercial van chassis, with hydraulic drive and electric steering, and was used as a testbed for integrating perception and navigation capabilities [THTS88]. The Mars Rover project at Jet Propulsion Laboratory aimed at developing the capabilities in machine intelligence systems required for a semi-autonomous vehicle to be used in remote planetary exploration [O'H73, Tho77, Ran86]. There are many other projects dealing with autonomous mobile robots, and additional references can be found in [SE87, CT90].

1.7.2 Major Research Issues

In spite of the diversity of the configurations and objectives of the mobile robots in these projects, all autonomous mobile robots must perform certain common functions. For very simple tasks, motion control and position localization functions are sufficient. Robots performing sophisticated tasks must also be able to perceive their surrounding environment, match sensing data with an internal world model, and build maps. To perform successfully in the real world, they must also deal with uncertainties in sensing and control. The various mobile robot projects contributed in identifying key navigation problems, and in producing approaches to solve such basic problems as localization, world-modeling/map-building, and dealing with uncertainties.

The localization problem is to determine the robot's position in some reference coordinate frame. For example, this may be necessary to determine the remaining distance to the goal or to construct a map of an incompletely known environment. Common techniques include using reference beacons [CL85], inertial navigation systems [Tho77], dead reckoning [Nil69], landmark recognition [Har85], and map matching [LK85].

World-modeling/map-building aims at constructing and maintaining a spatial description of the robot's environment (or updating a priori knowledge) using sensory data. For example, path planning involves the use of a map. The level of detail represented in a map is constrained not only by the available sensing accuracy, but also by considerations of cost such as storage requirements and processing complexity. Existing techniques use stochastic representations [SSC88], hierarchical representations [Elf85], symbolic representations [KTB87] and combinations of these [Gir84].

Uncertainty in mobile robot navigation exists in sensing, control, and prior knowledge. Dealing with it is closely coupled with localization and map-building. Many techniques including Kalman filtering [May79] and multi-sensor integration (sensory data fusion) [DW87] are used to reduce uncertainty [AF89, Cro89]. [LL92] presents a guaranteed navigation strategy under both sensing and control uncertainties. [TL92] describes navigation strategies to reduce sensing uncertainties.

Besides the above basic problems, mobile robot navigation involves other problems such as kinematics, motion control and trajectory generation. However, these problems tend to be tightly coupled with the specific configuration of a robot, and they are beyond the scope of our concern.

1.7.3 Dealing with Unexpected Obstacles

Approaches to robot navigation with no prior knowledge have been proposed by Khatib [Kha86] and Lumelsky [LS86]. The only available information assumed by these approaches is the robot's own configuration at any given moment, the goal configuration and the obstacles detected by the robot's sensors.

The potential field method proposed by Khatib places an artificial repulsive potential field around the obstacles and an attractive potential field at the goal. The motion along the steepest descent of the total potential field takes the robot towards the goal while avoiding obstacles. This method has been demonstrated to work well in practice when the environment is not densely occupied by obstacles. However, the total potential may have local minima, and the robot may get trapped at one of them. This shortcoming comes from the fact that the method relies only on local information about obstacles.

The boundary-following method proposed by Lumelsky uses three basic types of movements of the robot: move towards the goal on a straight line; move along the obstacle boundary in a predetermined direction (e.g., left or right); stop at the goal. Under this method, the robot moves towards its goal until the path is blocked by an obstacle, and then it moves around the obstacle boundary until the path towards the goal is clear. The method keeps track of when and where the robot starts to follow/leave the boundaries of obstacles, and uses the information in order to prevent infinite loops around the obstacles. Unlike the potential field method, this method is guaranteed to converge toward the goal. However, the path traveled by the robot tends to be far from optimal because the robot may have to explore large subsets of the obstacle boundaries on its way to the goal.

Numerous attempts have been made to improve the performance of these two basic approaches [Ark87, Kro84, KV89, Sla90]. Yet, they showed only marginal successes in gross navigation. Successful gross navigation requires not only knowledge of the robot's local surroundings, but also knowledge about places that are beyond the robot's local surroundings. The two levels of knowledge (i.e., local and global knowledge) lead to a natural division of the navigation system into two smaller interacting components: global planning and local reaction.

An approach combining a planning and a reaction component has previously been proposed by Krogh and Thorpe [KT86]. In this approach, a sequence of critical points along a globally desirable path are first computed. Potential fields are then used for local feedback to drive the robot along a collision-free path using the critical points as subgoals. At execution time, the navigation system has no model (similar to our channel) of the expected free space. Hence, it does not use a potential depending on this knowledge. As a result, it may run into trouble when critical points are occupied or hidden by unexpected obstacles. Accidental arrangements of unexpected obstacles can also take the robot far away from the desirable path.

The channel navigation technique described in this dissertation is an extension of this approach. The multi-layered architecture of our navigation system is similar to the subsumption architecture in a sense that each layer deals the event at a different level of competence. The difference is that the layers in our system architecture make use of the global information as well as the local information of the robot's workspace, and thus ensure better global behavior.

1.8 Summary of Results and Contributions

This thesis investigates mobile robot navigation in a stationary indoor environment in the presence of unexpected obstacles. It proposes the design of a new navigation system that interweaves planning and reaction components. This system has been implemented. Experimentation has been conducted successfully with simulated and real robots.

Our results bring two levels of contribution: (1) the system architecture itself and (2) the planning/reaction techniques embedded in this architecture.

The proposed architecture interweaves planning and reaction. In itself, this idea is not new. It has previously been developed in the artificial intelligence community, but for high levels of reasoning in task performance. It has not yet been thoroughly investigated at the robot navigation level, where geometry plays a key role. Our combination of planning and reaction is based on two new key ideas:

- The reaction component must have some global knowledge of the robot's workspace in order to react appropriately to unexpected obstacle events. This knowledge can be provided in the form of a lesser-committed motion plan generated by a planning component aware that unexpected obstacles may be present in the workspace. In our system, lesser-committed plans take two forms: channels made up of adjacent parallelepipedic cells and valley-shaped potentials.

- Multiple layers of treatment deal with classes of events according to their expected frequency. The top layer can treat alone all the events that the lower levels are intended for. The lower levels only provide more efficient treatment. This architecture makes it possible to introduce a reliable function (treatment of unexpected obstacles in our case) by building the top layer. This function can then be made more efficient by adding new layers. Our system consists of three layers: channel planning, local replanning and global replanning.

At the technical level, our work brings the following contributions:

- It introduces the concept of a channel as a lesser-commitment motion plan, and instantiates this concept as a sequence of parallelepipedic cells that can easily be generated using an approximate cell decomposition planning method.

- It defines potential field functions computed on-line to navigate in a channel toward the goal and simultaneously react to unexpected obstacles. The potential is guaranteed to be local-minima free when there are no unexpected obstacles.

- It presents a new way to escape local minima on-line by replanning a local path and integrating it in the current potential field function using the notion of a valley-shaped potential. This technique is general and can be used in other potential-field-based navigation systems.

These contributions are supported by experiments with the implemented navigation system.

1.9 Thesis Outline

The rest of this thesis is organized as follows:

Chapter 2 describes the internal structure of a channel and its generation from the given geometric model of the robot environment.

Chapter 3 defines a local-minima-free potential field in the channel, when there is no unexpected obstacle. It extends this potential field to the case with unexpected obstacles. It shows how this potential is used to compute the motion commands during navigation.

In Chapters 4 and 5 we discuss local replanning and global replanning. Chapter 4 addresses the problem of recognizing local minima and proposes a local planning technique to escape them within a cell of the channel. Chapter 5 extends this technique to multiple cells to deal with the case where local replanning fails to find an escape route within a single cell. It also presents a global replanning method to deal with the case where a channel is completely obstructed by unexpected obstacles.

In Chapter 6 we present the experiments done with our navigation system. This chapter describes three implementations: computer simulation, Robotworld, and GOFER. It discusses, based on the results with the simulated robot system, the sensitivity of the navigation system to sensing uncertainties. Also, in the case of the mobile robot GOFER, it discusses the impact of nonholonomic kinematic constraints on the navigation system.

Chapter 7 summarizes our work, comments on the limitations of the approach

and techniques presented in this dissertation, proposes a number of extensions to the current work, and provides a guide/suggestion for future research.

Appendices consists of three parts. Appendix A shows two additional examples of navigation in channel, which is described in Chapter 3. Appendix B describes GOFER's hardware in detail. Appendix C shows three simulation results of GOFER navigation in the presence of no unexpected obstacles, unexpected obstacles with no local minima, and unexpected obstacles with a local minimum, respectively.

Chapter 2

Channel Definition and Generation

The planning component of our navigation system generates a channel, i.e., a set of contiguous paths, using the global information of known obstacles. A channel provides guidance for the robot from the initial to the goal configuration, while leaving significant freedom of choice to the reaction component to deal with unexpected obstacles, and allowing the fast computation of motion commands. In this chapter we instantiate this concept into a specific representation and we present a method for computing it.

2.1 Configuration Space

Let us denote the robot by $\mathcal{A}$ and the workspace by $\mathcal{W}$. We attach a Cartesian frame $\mathcal{F}_\mathcal{A}$ to $\mathcal{A}$ and a Cartesian frame $\mathcal{F}_\mathcal{W}$ to $\mathcal{W}$. See Figure 2.1. A **configuration** q of $\mathcal{A}$ is a specification of the position and orientation of $\mathcal{F}_\mathcal{A}$ with respect to $\mathcal{F}_\mathcal{W}$. The **configuration space** of $\mathcal{A}$, denoted by $\mathcal{C}$, is the set of all the possible configurations of $\mathcal{A}$. The subset of $\mathcal{W}$ occupied by $\mathcal{A}$ at configuration q is denoted by $\mathcal{A}(q)$.

Throughout this report, we model the robot as a two-dimensional object moving in a planar workspace $\mathcal{W}$ isomorphic to $\mathbf{R}^2$. This corresponds to projecting both the

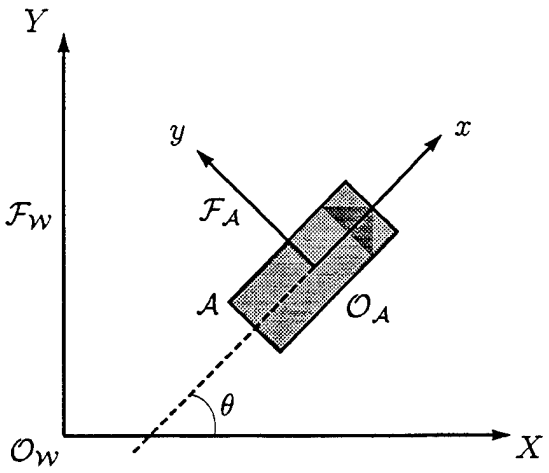


Figure 2.1: Robot and workspace Cartesian frames

real robot and the obstacles into the horizontal ground. Therefore, $\mathcal{C}$ is isomorphic to either $\mathbf{R}^2$ (if $\mathcal{A}$ can only translate or if it is a disc) or $\mathbf{R}^2 \times \mathcal{S}^1$, where $\mathcal{S}^1$ is the unit circle (if $\mathcal{A}$ can both translate and rotate). However, since the concepts underlying our approach are more general,[1] we keep our presentation as independent as possible from these assumptions. In general, $\mathcal{C}$ is a manifold of some dimension m.

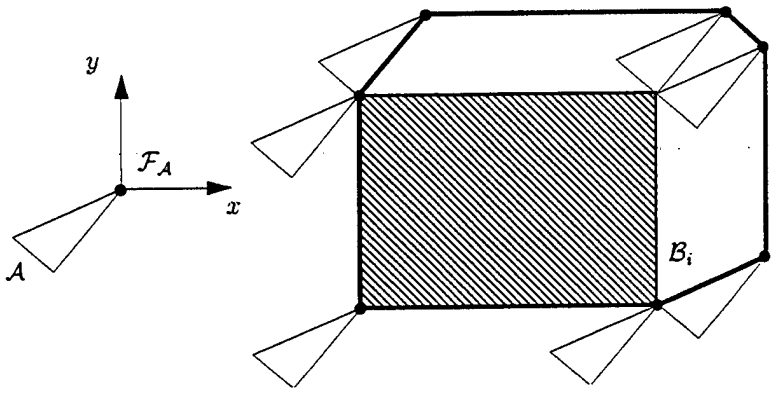
In addition to $\mathcal{A}$, $\mathcal{W}$ contains known stationary obstacles denoted by $\mathcal{B}_i, i = 1, \ldots, n$. Each obstacle $\mathcal{B}_i$ maps into $\mathcal{C}$ to the subset $\mathcal{CB}_i$ of configurations where $\mathcal{A}$ intersects $\mathcal{B}_i$. This subset is defined by [LP83, Lat91]:

$$\mathcal{CB}_i = \{\boldsymbol{q} \in \mathcal{C} \mid \mathcal{A}(\boldsymbol{q}) \cap \mathcal{B}_i \neq \emptyset\}.$$

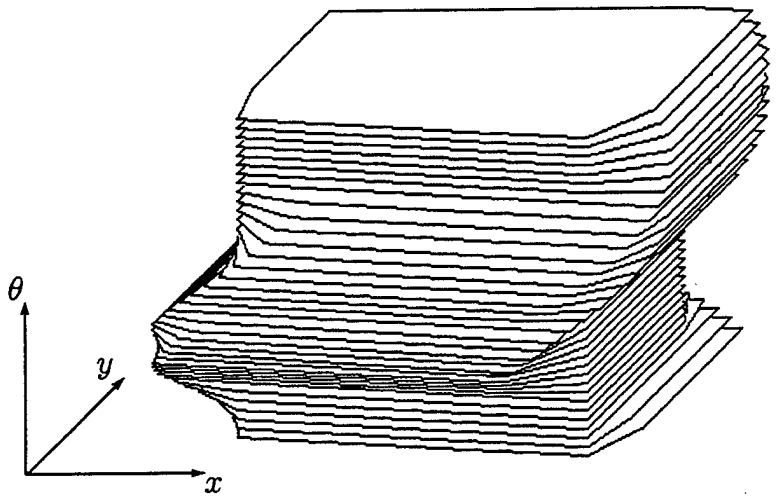
It is called a **$\mathcal{C}$-obstacle**. Figure 2.2 shows $\mathcal{C}$-obstacles corresponding to a rectangular obstacle $\mathcal{B}_i$ and a triangular robot $\mathcal{A}$. In (a), $\mathcal{A}$ has a fixed orientation, and the $\mathcal{C}$-obstacle is a polygon in the robot's 2-dimensional configuration space. In (b), the orientation θ of $\mathcal{A}$ can vary in $[0, 2\pi]$, and the $\mathcal{C}$-obstacle is a volume bounded by patches of ruled surfaces in the robot's 3-dimensional configuration space.

The $\mathcal{C}$-obstacle region is $\mathcal{CB} = \bigcup_{i=1}^{n} \mathcal{CB}_i$. The complement to the $\mathcal{C}$-obstacle region

[1]For example, the same approach could be extended to a six-degree-of-freedom free-flying platform or to an articulated manipulator arm.



(a)



(b)

Figure 2.2: $\mathcal{C}$-obstacles in 2 and 3 dimensions

in $\mathcal{C}$ is called the **free** space and is denoted by $\mathcal{C}_{free}$:

$$\mathcal{C}_{free} = \mathcal{C} \backslash \mathcal{CB}.$$

A **path** of the robot $\mathcal{A}$ from the initial configuration q_{init} to the goal configuration q_{goal} is a continuous map $\tau : [0,1] \rightarrow \mathcal{C}$, with $\tau(0) = q_{init}$ and $\tau(1) = q_{goal}$. A **collision-free path** is any continuous map $\tau : [0,1] \rightarrow \mathcal{C}_{free}$.

We will introduce unexpected obstacles later.

2.2 Definition of a Channel

We parametrize a configuration $q \in \mathcal{C}$ by a list of m generalized coordinates $(q_1, \ldots, q_m)$ in a Cartesian space, where m is the dimension of the configuration space manifold. We assume, without practical loss of generality, that the range of possible values for the $q_i's$ are closed intervals $[q_i^{min}, q_i^{max}]$. Hence, we represent $\mathcal{C}$ as a closed rectangloid:

$$[q_1^{min}, q_1^{max}] \times \ldots \times [q_m^{min}, q_m^{max}] \subset \mathbf{R}^m.$$

For our mobile robot example, if $\mathcal{C} = \mathbf{R}^2$, we take $q = (x, y)$, with x and y being the coordinates of the origin $\mathcal{O}_\mathcal{A}$ of $\mathcal{F}_\mathcal{A}$ with respect to $\mathcal{F}_\mathcal{W}$. If $\mathcal{C} = \mathbf{R}^2 \times \mathcal{S}^1$, we take $q = (x, y, \theta)$, with x and y defined in the same fashion, and $\theta \in [0, 2\pi]$ being the angle (*modulo* 2π) between the x-axes of $\mathcal{F}_\mathcal{W}$ and $\mathcal{F}_\mathcal{A}$. We represent $\mathcal{C}$ as $[x^{min}, x^{max}] \times [y^{min}, y^{max}] \times [0, 2\pi]$ with the faces $\theta = 0$ and $\theta = 2\pi$ made identical.

We formally define a channel as a sequence $(\kappa_1, ..., \kappa_p)$, $p \geq 1$, of rectangloids, called **cells**, such that:

1. $q_{init} \in \kappa_1$ and $q_{goal} \in \kappa_p$ – i.e., the channel connects the initial configuration to the goal configuration;

2. $\forall i \in [1, p], int(\kappa_i) \subset \mathcal{C}_{free}$ – i.e., the interior of every cell is contained in $\mathcal{C}_{free}$;

3. $\forall i, j \in [1, p], i \neq j, int(\kappa_i) \cap int(\kappa_j) = \emptyset$ – i.e., no two cells overlap;

4. $\forall i \in [1, p-1], \kappa_i \cap \kappa_{i+1}$ is an $(m-1)$-dimensional rectangloid – i.e., two successive cells in the sequence are adjacent by sharing a portion of their boundary having non-zero measure in $\mathbf{R}^{m-1}$.

The $(m-1)$-dimensional rectangloid $\kappa_{i-1} \cap \kappa_i$ (resp., $\kappa_i \cap \kappa_{i+1}$) is called the **access gate** (resp., the **exit gate**) of κ_i, with the convention that $\kappa_0 = \kappa_{p+1} = \emptyset$.

When $\mathcal{C} = \mathbf{R}^2 \times \mathcal{S}^1$, a spurious effect of the Cartesian representation of the configuration space is to introduce an artificial boundary for the cells at $\theta = 0$ and $\theta = 2\pi$. We remove this artificial boundary by considering two regions of the following forms:

$$[x^{min}, x^{max}] \times [y^{min}, y^{max}] \times [0, \theta_1]$$

and

$$[x^{min}, x^{max}] \times [y^{min}, y^{max}] \times [\theta_2, 2\pi],$$

as a single cell, although it is represented as two rectangloids.

If $\mathcal{C} = \mathbf{R}^2$, the boundary $\partial\kappa_i$ of a cell κ_i simply consists of the four edges of the cell. If $\mathcal{C} = \mathbf{R}^2 \times [0, 2\pi]$, $\partial\kappa_i$ consists of the six faces of the corresponding rectangloid, if κ_i does not range over all the orientations in $[0, 2\pi]$. Otherwise, it only consists of the four faces perpendicular to the x and y axes. The **boundary** $\partial\Pi$ of the channel $\Pi = (\kappa_1, \ldots, \kappa_p)$ is defined as:

$$\partial\Pi = \bigcup_{i=1}^{p} \partial\kappa_i - \bigcup_{i=1}^{p-1} \kappa_i \cap \kappa_{i+1}$$

2.3 Construction of a Channel

A channel is constructed by iteratively partitioning the Cartesian representation of the configuration space $\mathcal{C}$ into non-overlapping rectangloid cells parallel to the coordinate axes. At each iteration, every cell is labelled as **empty**, **full**, or **mixed**, depending on whether it has no intersection with the $\mathcal{C}$-obstacle region $\mathcal{CB}$, it is completely

contained in $\mathcal{CB}$, or only partially contained in $\mathcal{CB}$.

The connectivity graph representing the adjacency relation among the empty and mixed cells is constructed and searched for a sequence of cells, $(\kappa_1, ..., \kappa_p)$, $p \geq 1$, such that $q_{init} \in \kappa_1$, $q_{goal} \in \kappa_p$, and $\forall i \in [1, p-1], \kappa_i$ and κ_{i+1} are adjacent. If a sequence containing only empty cells is found, it is the generated channel. If a sequence is found, but contains mixed cells, those cells are decomposed into smaller cells, and the connectivity graph is updated and searched again. This iterative process is bounded by setting a minimal size on the mixed cells. This size may also be used to take uncertainty in robot control into account.

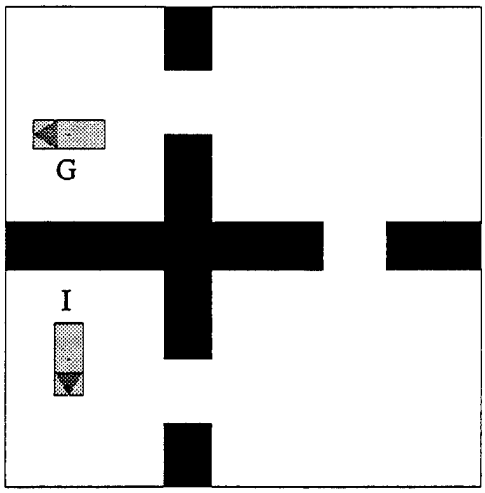
The actual decomposition and search techniques used in our navigation system to generate channels are described in [Zhu92]. They are resolution-complete, i.e.: if a channel exists, the planner is guaranteed to return one in a finite amount of time, provided that the minimal size of a mixed cell is set sufficiently small; if no channel exists or the minimal size of a mixed cell is too big, the planner returns failure in a finite amount of time. During channel generation, most of the computation time is spent in decomposing the configuration space and labelling the cells. However, if the model of the environment is not changed often, most of the decomposition and labelling work can be saved for future use, so that channel generation rapidly reduces to searching a preexisting connectivity graph.

2.4 Examples

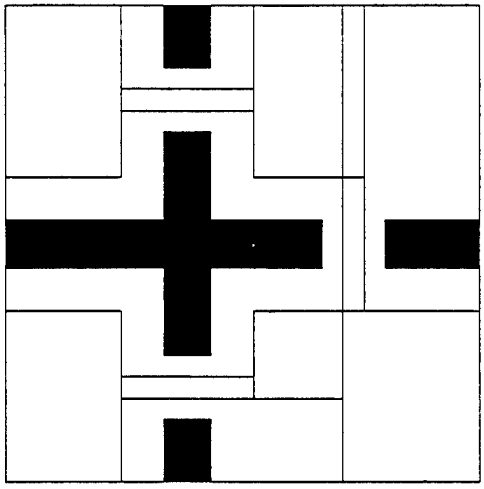
Figure 2.3 shows an example of the robot's workspace and its corresponding channel. (a) shows a robot's workspace with known obstacles and the initial and goal configurations of the robot. (b) shows the 2-dimensional projection of the channels into the workspace plane. This projection is an incomplete, but much more readable representation of the complete 3-dimensional channel shown in (c). We will exclusively use this representation in the rest of this report, unless the complete perspective view is absolutely needed.

Figure 2.4 shows two more examples. Note that both the channels in Figure 2.3

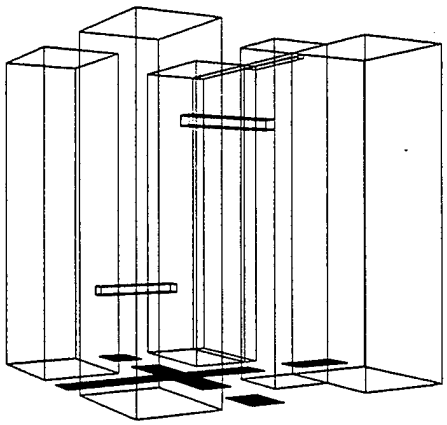
(c) and 2.4 (f) contain cells whose ranges along the θ-axis is a subset of $[0, 2\pi]$.



(a)



(b)



(c)

Figure 2.3: Typical example of a workspace and a channel

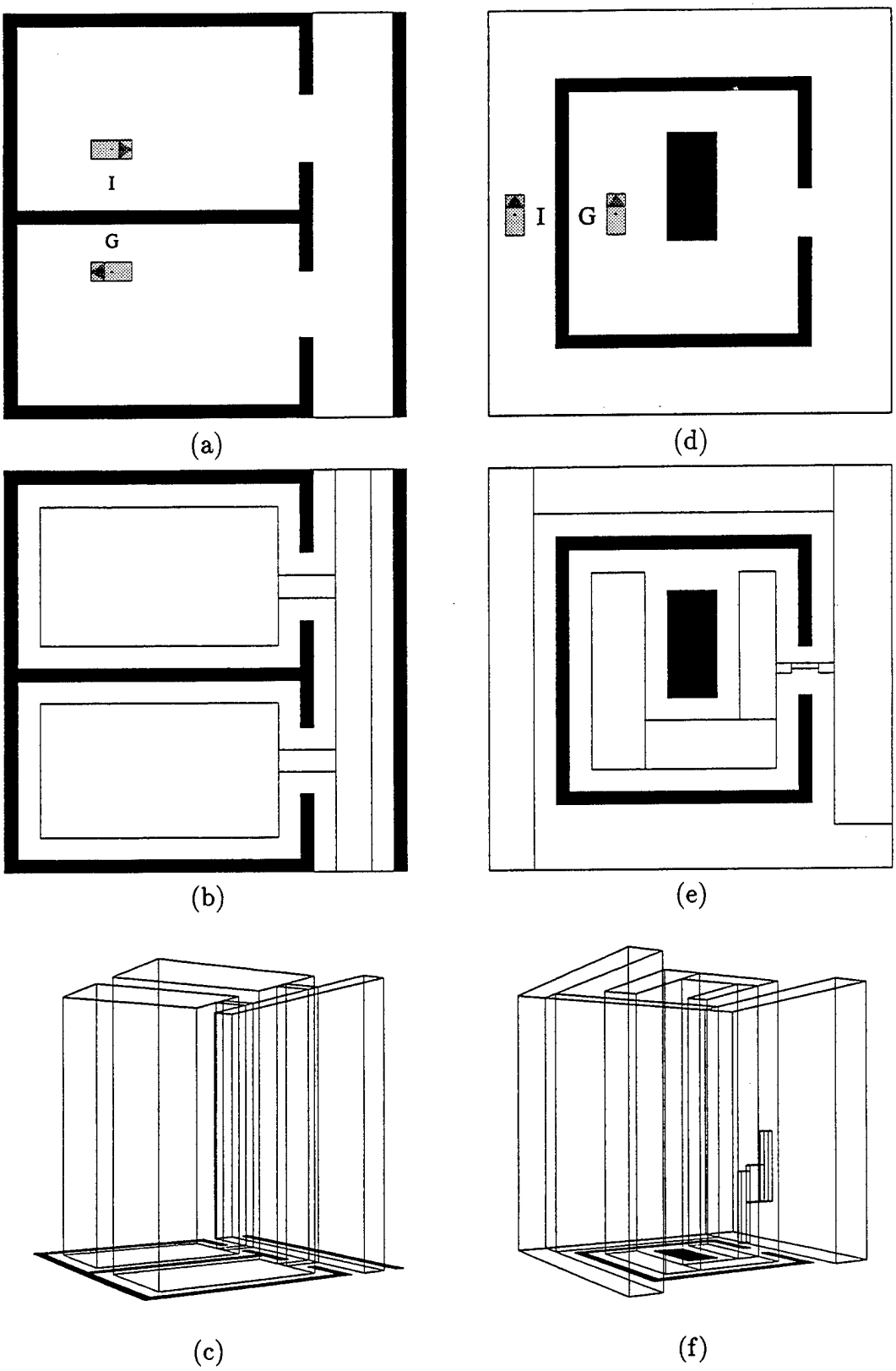


Figure 2.4: More channel examples

Chapter 3

Navigation in Channel

Given a channel, the reaction component generates motion commands for the robot by defining an artificial potential field $\mathbf{U}$ over the channel and tracking its negated gradient $-\nabla\mathbf{U}$. $\mathbf{U}$ should have a global minimum at the goal configuration q_{goal}. It should grow towards infinity when the distance between the robot and an unexpected obstacle (if any) tends towards 0, in order to avoid collisions. It should also grow to infinity when the robot's configuration tends towards the channel's boundary, in order to keep the robot within the channel. We would also very much like $\mathbf{U}$ to have no local minima. However, since the values of this function depends on sensory data triggered by unexpected obstacles, there seems to be no way to guarantee this last property.[1]

The potential $\mathbf{U}$ is constructed at every configuration q as the sum of two functions:

$$\mathbf{U}(q) = \mathbf{U}_c(q) + \mathbf{U}_s(q)$$

where $\mathbf{U}_c(q)$ is a function of both the robot's current configuration and the geometry of the channel. $\mathbf{U}_s(q)$ is a function of the current configuration relative to unexpected obstacles detected by sensors. $\mathbf{U}_c$ is called the **channel potential**. $\mathbf{U}_s$ is called the

[1]Both analytical and numerical techniques have been developed for computing local-minima-free potentials [RK90, BL91]. But these techniques assume complete prior knowledge of the obstacles; hence, they are not applicable here.

unexpected-obstacle potential. $\mathbf{U}_c$ is free of local minima. Since both $\mathbf{U}_c(q)$ and $\mathbf{U}_s(q)$ depend on the robot's configuration q, it is very important to have accurate estimates of q (i.e., localization) throughout the navigation. The localization is typically obtained by dead reckoning and/or environment sensing.

3.1 Channel Potential

One simple way to construct $\mathbf{U}_c$ is to add an attractive potential pulling the robot toward its goal and a repulsive potential pushing it away from the channel's boundary. However, because a channel is usually a non-convex region, this simple construction could result in a function $\mathbf{U}_c$ with local minima. This is not acceptable, since the robot could get stuck at one of them even in the absence of unexpected obstacles. The local-minima-free potentials proposed in [RK90, BL91] can be applied to this non-convex channel, but their computation is too complex to be done in real time because they are based on global information. To overcome this problem, we patch together several potential functions. Each such function is defined in a rectangloid for an intermediate goal selected in the channel and has only one minimum at this intermediate goal. The definition domain of the various potential functions overlap so that the robot can shift from one function to the next before it attains an intermediate goal (hence, avoiding the corresponding minimum).

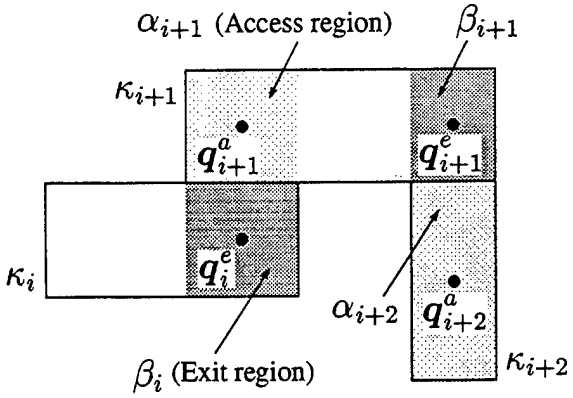
3.1.1 Intermediate Goals

We construct a sequence of intermediate goal configurations in the channel, the last one being q_{goal}. Then we define $\mathbf{U}_c$ piecewisely so that the robot $\mathcal{A}$ is successively attracted by each intermediate goal configuration. The issues are:

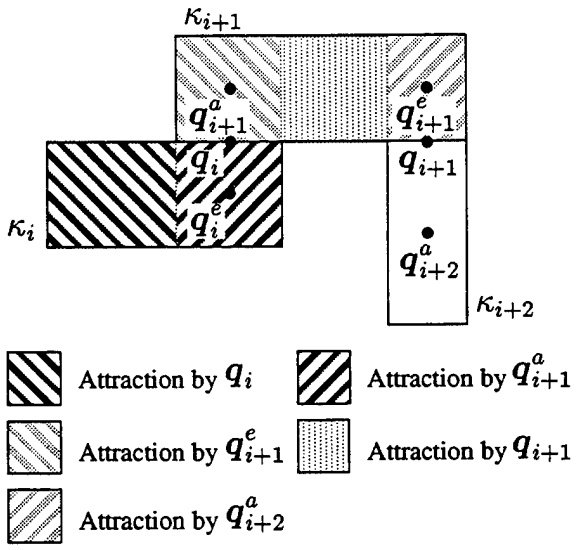
1. How to choose the intermediate goal configurations?

2. When to shift from one intermediate goal to the next?

A possible sequence of intermediate goal is $(q_1, \ldots, q_p)$, where q_i is the midpoint of $\kappa_i \cap \kappa_{i+1}$, for $i = 1, \ldots, p-1$, and $q_p = q_{goal}$. Then, in each cell κ_i, we can



(a) Intermediate goals in access and exit regions



(b) Goal shifting between regions

Figure 3.1: Intermediate goals

define the potential $\mathbf{U}_c^i, (i = 1, \ldots, p)$ as the sum of an attractive potential pulling $\mathcal{A}$ toward q_i and a repulsive potential pushing it away from the boundary of the channel restricted to the cell, i.e., $\partial\kappa_i \cap \partial\Pi$. When $\mathcal{A}$ traverses $\kappa_i \cap \kappa_{i+1}$, the attractive goal is switched to q_{i+1}. When the last cell, κ_p, is entered, q_{goal} becomes the goal.

The problem with this definition is that the attractive potential has a minimum at each intermediate goal q_i. Therefore, in the vicinity of each q_i, $\mathcal{A}$ is mainly under the influence of the repulsive potential, if any, and, at best, lacks any goal-oriented behavior. This is likely to increase the risks of spurious stable equilibrium states near the intermediate goals. A solution to this drawback is to make $\mathcal{A}$ abandon every intermediate goal before it is attained and shift to the next, so that the attractive force never vanishes. This led us to retain a slightly different definition for $\mathbf{U}_c$, which we present below.

Let us consider a cell κ_i. We denote by α_i (resp., β_i) the region obtained by sweeping the access gate (resp., the exit gate) of κ_i perpendicularly to itself inside κ_i. Both α_i and β_i may be identical to κ_i. In the first cell of the channel, we only sweep its exit gate (β_1). In the last cell, we only sweep its access gate (α_p). Figure 3.1 (a) illustrates the construction of the α_i's and the β_i's in a 2-dimensional channel.

For every cell κ_i, we define the midpoints of α_i and β_i as two additional intermediate goal configurations, which we respectively denote by q_i^a and q_i^e.[2] The sequence of intermediate goals is:

$$(q_1^e, q_1, q_2^a, q_2^e, q_2, \ldots, q_{p-1}^e, q_{p-1}, q_p^a, q_{goal}).$$

These intermediate goals are also shown in Figure 3.1 (a). For some cells, it is possible that q_i^a and q_i^e coincide. If two intermediate goals coincide, they are treated as a single one.

[2]Using the midpoints as intermediate goals may result in an inefficient path when the neighboring cells in the channel have "big" differences in their shapes and sizes. In the actual implementation, we take a point on the line segment connecting the midpoints of the access (resp., exit) region and the midpoints of $\kappa_{i-1} \cap \kappa_i$ (resp., $\kappa_i \cap \kappa_{i+1}$). The exact location depends on the relative size of the neighboring cells.

3.1.2 Intermediate-Goal Potential

We construct $\mathbf{U}_c$ in a piecewise fashion over overlapping rectangloid regions. Each such region is either an **access rectangloid**, i.e., α_i, an **exit rectangloid**, i.e., β_i, or a **regular rectangloid**, i.e., the complement of $\alpha_i \cup \beta_i$ in a cell κ_i, with $i \in [1, p]$. For every $i \neq p$, the goal configuration in the access rectangloid α_i is $\boldsymbol{q}_i^e$; the goal configuration in the regular rectangloid is $\boldsymbol{q}_i$; the goal configuration in the exit rectangloid β_i is $\boldsymbol{q}_{i+1}^a$. When $i = p$, the goal configuration is $\boldsymbol{q}_{goal}$ over all the cell κ_p.

$\mathbf{U}_c$ is defined over each rectangloid as the sum of two terms, an attractive term $\mathbf{U}_c^g$, which pulls the robot toward the goal configuration $\boldsymbol{q}_g$ of the rectangloid, and a repulsive term $\mathbf{U}_c^b$, which pushes the robot away from the boundary of the channel. $\mathbf{U}_c^g$ shifts its goal from one intermediate goal to the next whenever $\mathcal{A}$'s configuration enters a new (regular, access or exit) rectangloid. This is illustrated in Figure 3.1 (b). If $\mathcal{A}$'s configuration is in κ_i, $i \in [1, p-1]$, and not in $\alpha_i \cup \beta_i$, the current goal is $\boldsymbol{q}_i$. As soon as it enters β_i, the current goal becomes $\boldsymbol{q}_{i+1}^a$. When it enters κ_{i+1} (i.e., α_{i+1}), the current goal becomes $\boldsymbol{q}_{i+1}^e$, if $i + 1 \neq p$. Finally, if it enters the goal cell, the current goal becomes $\boldsymbol{q}_{goal}$. If $\mathcal{A}$'s configuration is in $\alpha_i \cap \beta_i$ ($\neq \emptyset$), β_i has a higher priority and the current goal is $\boldsymbol{q}_{i+1}^a$. If $p = 1$, there is no intermediate goal and $\boldsymbol{q}_{goal}$ is immediately taken as the goal to attain.

Shifting from one intermediate goal to the next as explained above results in a discontinuity of the attractive force. Such discontinuity can be smoothed at the servo level. Another technique would consist of shifting continuously from an intermediate goal to the next, by making the goal vary along the line segment connecting them.

3.1.3 Formal Definition of Potential

The potentials $\mathbf{U}_c^g$ and $\mathbf{U}_c^b$ can be formally defined in several ways. Our definitions are directly inspired from those given in [Kha86]. We take:

$$\mathbf{U}_c^g(\boldsymbol{q}) = \frac{1}{2} K_g \rho_g^2(\boldsymbol{q})$$

and

$$\mathbf{U}_c^b(\boldsymbol{q}) = \begin{cases} \frac{1}{2}K_b(\frac{1}{\rho_b(\boldsymbol{q})} - \frac{1}{\rho_o})^2 & \text{if } \rho_b(\boldsymbol{q}) \leq \rho_o, \\ 0 & \text{if } \rho_b(\boldsymbol{q}) > \rho_o. \end{cases}$$

where

- K_g and K_b are scaling factors,
- $\rho_g(\boldsymbol{q})$ is the distance between $\boldsymbol{q}$ and the current goal $\boldsymbol{q}_g$,
- $\rho_b(\boldsymbol{q})$ is the distance from $\boldsymbol{q}$ to the boundary of the channel,
- ρ_o is the **distance of influence** of the channel boundary.

In our implementation, we only consider the two cases where $\mathcal{C} = \mathbf{R}^2$ and $\mathcal{C} = \mathbf{R}^2 \times S^1$. In the first case, the distances ρ_g and ρ_b are simply the Euclidean distances from $\boldsymbol{q} = (x, y)$ to $\boldsymbol{q}_g = (x_g, y_g)$, and from $\boldsymbol{q} = (x, y)$ to $\partial\Pi$ (the boundary of Π). In the second case, we compute the distance between $\boldsymbol{q}_1 = (x_1, y_1, \theta_1)$ and $\boldsymbol{q}_2 = (x_2, y_2, \theta_2)$ as:

$$d(\boldsymbol{q}_1, \boldsymbol{q}_2) = \left[(x_1 - x_2)^2 + (y_1 - y_2)^2 + r^2\, l^2(\theta_1, \theta_2)\right]^{\frac{1}{2}}$$

where r is a scaling factor that we take equal to the maximal distance between the origin of the frame $\mathcal{F}_\mathcal{A}$ attached to $\mathcal{A}$ and the boundary of $\mathcal{A}$. If the cell ranges over all orientations in $[0, 2\pi]$, we take:

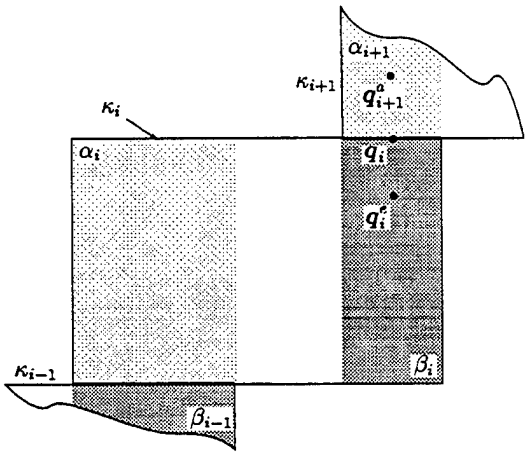
$$l(\theta_1, \theta_2) = \min(|\theta_1 - \theta_2|, 2\pi - |\theta_1 - \theta_2|).$$

If it ranges over a subset of $[0, 2\pi]$, we compute $l(\theta_1, \theta_2)$ as the length of the arc connecting the orientation θ_1 to the orientation θ_2 and contained in the angular range of the cell. We take:

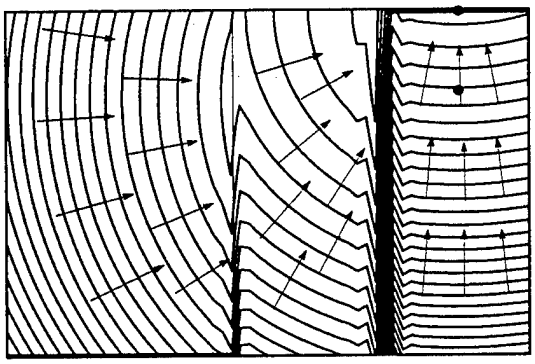
$$\rho_g(\boldsymbol{q}) = d(\boldsymbol{q}, \boldsymbol{q}_g)$$

and

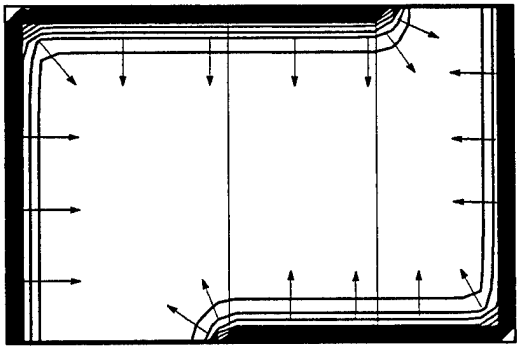
$$\rho_b(\boldsymbol{q}) = \min_{\boldsymbol{q}_b \in \partial\Pi} d(\boldsymbol{q}, \boldsymbol{q}_b).$$



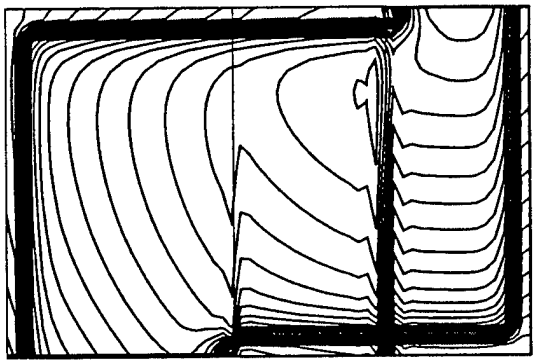
(a) Regions in a cell



(b) Attractive potential $\mathbf{U}_c^g$



(c) Repulsive potential $\mathbf{U}_c^b$



(d) Channel potential $\mathbf{U}_c = \mathbf{U}_c^g + \mathbf{U}_c^b$

Figure 3.2: Channel potential

Figure 3.2 shows the channel potential constructed using these definition in a 2-dimensional cell. Figure (a) shows the cell, its access (light gray) and exit (dark gray) regions, and the intermediate goals. (b) shows the equipotential contours of the goal potential in the cell. The arrows represent the negative gradients of the goal potential and, in each region, they point to the corresponding intermediate goal. (c) shows the equipotential contours of the wall potential. The distance of influence in the access (resp., exit) region is a fraction of the size of the access (resp., exit) gate in order to to prevent the creation of local minima of the channel potential. The wall potential vanishes near the center of these gates as well as in the central area of the cell. (d) shows the resulting channel potential.[3]

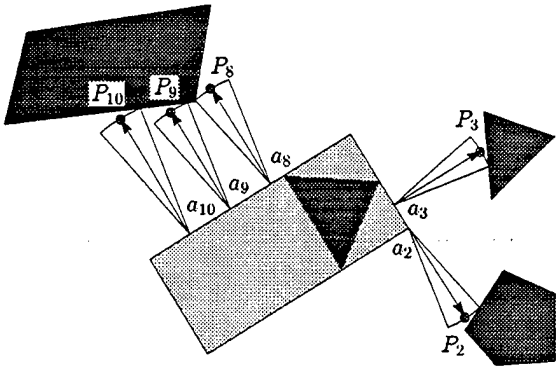
3.2 Unexpected-Obstacle Potential

3.2.1 Principle

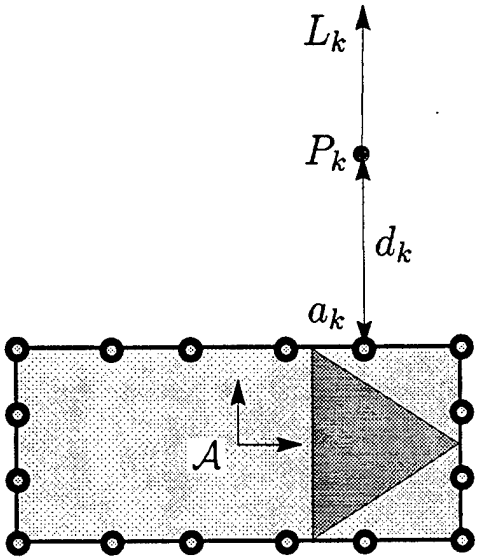
We now define the potential induced by unexpected obstacles. We assume that these unexpected obstacles are detected by N proximity range sensors mounted on the robot. We denote these sensors by $S_1, ..., S_N$. Typically, these proximity sensors are infra-red emitter/receiver pairs or sonars fixed on the boundary of the robot (see Figure 3.3 (a)).

At every instant, each sensor S_k, $k = 1, ..., \mathrm{N}$, measures the distance d_k from the point a_k in $\mathcal{A}$'s boundary, where S_k is located (see Figure 3.3 (b)) to an obstacle along a ray L_k fixed with respect to $\mathcal{A}$. (By convention, when S_k detects no obstacle, d_k becomes infinity.) This "perfect sensing" assumption may not be verified for a single measurement. However, the effect of a sensing error on the behavior of the robot is very brief, since another measurement will be repeated shortly after. Nevertheless, the impact of imperfect sensing on the performance of the navigation system will be discussed later in Chapter 6.

[3]The program used to draw the equipotential contours do not produce contours correctly near the boundary of two regions where values of the potential are discontinuous. The apparent local minima shown in Figures 3.2 (d), 3.4 (b), and 4.1 (b) are falsely generated due to this defect of the plotting program.



(a)



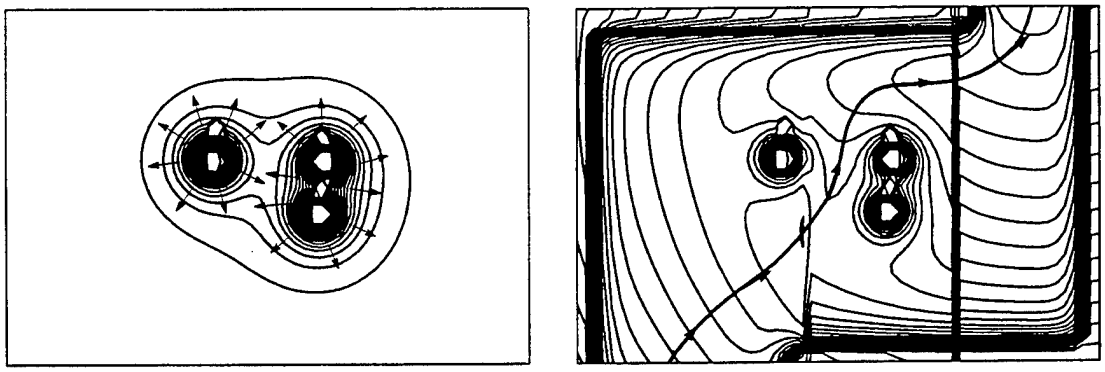
(b)

Figure 3.3: Detection of obstacles

Let P_k denote the point of $\mathcal{W}$ located at distance d_k of the boundary of $\mathcal{A}(\mathbf{q})$ along L_k. If $P_k \notin \bigcup_{i=1}^{n} \mathcal{B}_i$ (the region of the known obstacles), this point is called the **contingency** detected by S_k. At each instant, the contingencies are treated as independent point obstacles which create circular repulsive potential around them pushing the robot away from the unexpected obstacles. In order to be more consistent with the rest of the navigation system, we should rather test that the $\mathcal{C}$-obstacle corresponding to a point obstacle P_k intersects the channel, before calling it a contingency; otherwise, the robot motion may be unnecessarily affected without reducing its chance of success. This additional test can be relatively time-consuming, especially if there are many P_k's. We chose not to make this test.

We use proximity sensors as the source of information about unexpected obstacles. However, we could use other ranging devices instead. For example, a typical laser-camera range-finder projects a plane of light from the laser either through a cylindrical lens or by panning the laser beam around the robot, and the camera captures the light reflected by the obstacles' boundary. The range information on the detected obstacles' boundary is obtained through triangulation. Such a range-finder provides more complete and accurate information about the outlines of detected obstacles and thus would enhance the robustness of the navigation system. However, treating the detected obstacles as non-point geometric objects such as line segments or curved lines would require more processing time. As a compromise, a limited number of light-rays can be used to select point obstacles from the outlines of the detected obstacles. Then, the sensor model used in our navigation system can be applied as well.

We will see in the next chapter that the robot keeps track of the detected unexpected obstacles in a local model of the workspace associated with the channel. However, this model is not exploited here; only the current sensory data are used at each instant. This choice makes the computation of the potential not only faster, but also valid when there are moving unexpected obstacles. On the other hand, it may produce additional zero-force configurations (i.e., local minima).



(a) Unexpected-obstacle potential $\mathbf{U}_s$ (b) Total potential $\mathbf{U} = \mathbf{U}_c + \mathbf{U}_s$

Figure 3.4: Unexpected-obstacle potential and total potential

3.2.2 Formal Definition

The unexpected-obstacle potential is constructed in two steps. First, a potential function $\mathbf{V}_k$, called **contingency potential**, is defined *over the workspace* for every contingency point P_k detected by the sensors. This definition, given below, is very similar to the definition of the channel potential, but in a different space. In the second step, the various contingency potentials are combined into a function defined over the configuration space. This function is the unexpected-obstacle potential $\mathbf{U}_s$.

- If $P_k \in \bigcup_{i=1}^{n} \mathcal{B}_i$, P_k is not a contingency. Then, $\mathbf{V}_k(x, y) = 0$ for all $(x, y) \in \mathbf{R}^2$.

- Otherwise:

$$\mathbf{V}_k(x,y) = \begin{cases} \frac{1}{2}K_s(\frac{1}{\rho_k(x,y)} - \frac{1}{\rho'_0})^2 & \text{if } \rho_k(x,y) \leq \rho'_0, \\ 0 & \text{if } \rho_k(x,y) > \rho'_0. \end{cases}$$

where

- K_s is a scaling factor,
- $\rho_k(x, y)$ is the Euclidean distance (in $\mathbf{R}^2$) between the point (x, y) and P_k,
- ρ'_o is the distance of influence of an unexpected obstacle.

This potential induces a force field $G_k = -\nabla \mathbf{V}_k$ over the workspace, which only applies to the point a_k. The force G_k applied at a_k is converted to a generalized force

F_k as follows:

- If $\mathcal{C} = \mathbf{R}^2$, $F_k = G_k$.
- If $\mathcal{C} = \mathbf{R}^2 \times S^1$, F_k is a vector with three components. The first two components are those of G_k. The third one is the outer product[4] $O_A a_k \times G_k$.

One can easily verify that $F_k(\mathbf{q}) = -\nabla \mathbf{U}_k(\mathbf{q})$, where $\mathbf{U}_k(\mathbf{q}) = \mathbf{V}_k(a_k(\mathbf{q}))$ [Lat91]. Therefore, $\mathbf{U}_s(q)$ is defined as:

$$\mathbf{U}_s(q) = \sum_{i=1}^{N} \mathbf{U}_k(q) = \sum_{i=1}^{N} \mathbf{V}_k(a_k(\mathbf{q})).$$

Figure 3.4 shows the unexpected-obstacle potential for a point robot. Figure (a) shows the equipotential contours around three detected point obstacles. To simplify the illustration, we use a point robot, but, as mentioned above, this construction extends to a non-point robot in 2 and 3 dimensions. (b) shows the total potential that is obtained by by adding the channel potential of Figure 3.2 and the unexpected-obstacle potential of (a). It also shows a path following the negative gradient of the total potential, thus illustrating the navigation of the robot in the presence of the detected point obstacles.

The distance of influence ρ_o (channel boundary) (ρ_o) and ρ'_o (unexpected obstacle) vary through the channel, but are constant over each cell (except, possibly the last cell). They increase with the size of the cell and the size of the exit gate. In the last cell, they are taken small enough so that $\mathbf{U}_c^b$ and $\mathbf{U}_s$ become zero at q_{goal} in order for the robot to reach q_{goal} even if an unexpected obstacle lies close to the robot at its goal configuration. In fact, in the last cell, they can be gradually reduced as the robot gets closer to q_{goal}.

[4]In order for the outer product to be non-zero, the line supporting L_k should not pass through O_A. This suggests that on non-circular mobile robots, range sensors should not be distributed along a ring.

3.3 Computation of Motion Commands

The general dynamic equation of $\mathcal{A}$ in configuration space is:[5]

$$\Lambda\ddot{q} + \mu(q, \dot{q}) - F_m = 0$$

where $\dot{q}$ and $\ddot{q}$ are the generalized velocity and acceleration of the robot, Λ is the kinetic energy matrix, $\mu(q, \dot{q})$ is the centrifugal and Coriolis generalized forces, and F_m is the generalized force applied by the actuators. The negative gradient of the potential $\mathbf{U}$ can be used as an external force acting on $\mathcal{A}$ to prescribe its behavior such that $\mathcal{A}$ is treated as a unit-mass particle moving under the influence of $\mathbf{U}$. Thus, the motion command F_m can be computed as [Kha86]:

$$F_m = \Lambda[-\nabla\mathbf{U}(\mathbf{q})] + \mu(q, \dot{q}).$$

Such a motion command compensates the dynamic effects while it achieves the desired behavior of $\mathcal{A}$ specified by $\mathbf{U}$. In practice, however, the speed of a mobile robot is relatively slow, and therefore, the dynamic effects of the robot motion can be ignored. Specifying the desired acceleration to the (kinematic) controller of the robot is usually sufficient. For instance, GOFER is built on a mobile base with a built-in PID controller. It provides a position control mode and a speed control mode. In a position control mode, we can specify a desired position (and orientation), and assign arbitrary values (within limited ranges) for the velocity and acceleration. In a speed control mode, which is the control mode used for our navigation system, a desired velocity of the robot is specified. The motion command for such a robot can be computed directly from $\mathbf{U}$ as:

$$\dot{q}_{des} = K_m[-\nabla\mathbf{U}(\mathbf{q})]$$

where K_m is a scaling gain. K_m is selected such that the maximal speed of the robot

[5]When the robot is a wheeled vehicle moving on a plane, there exist frictions between the wheels and the plane. Then, the equation of motion is $\Lambda\ddot{q} + \mu(q, \dot{q}) - F_{fric} - F_m = 0$ where F_{fric} is the generalized friction force.

does not exceed some prespecified value. The desired acceleration is also specified not to exceed the actuator limit.

Using the control scheme described in [Kha86], we introduce a damping term proportional to the velocity in the control of the robot. The damping term makes the robot decelerate when it gets close to an intermediate goal. This happens only at the final goal and at places where the channel is narrow and/or winding. Practically, experimentations show that, except for short acceleration and deceleration segments, the robot navigates at the selected maximal speed.

3.4 Examples

Figure 3.5 shows both the intermediate goals in the channel of Figure 2.3 (b) and the configurations where goal shifting occurs. In (a), the transparent robots placed at the intermediate goals in a 2-dimensional channel show the corresponding goal configurations; the initial and goal configurations are shown with the gray robots. Figures (b) through (i) show various configurations of the robot (gray) and the corresponding intermediate goal configurations (transparent).

Figure 3.6 (a) shows a path followed by the robot in the channel when there are no unexpected obstacles. Figure 3.6 (b) shows a path followed by the robot in the presence of unexpected obstacles (shown grey). In this example, 16 proximity-sensors were used to detect obstacles. Figure 3.7 illustrates the detection of unexpected obstacles along the path. In (b) and (d), the proximity sensors also detect known obstacles. The corresponding detected points are not contingencies, and, thus, should not contribute to the unexpected-obstacle potential. In practice, however, sensing is not perfect, and it is difficult (and time-consuming) to distinguish between the detected points from known obstacles and unexpected obstacles. Therefore, we relax the definition of contingencies to include the detected points from known obstacles as well as unexpected obstacles. This changes the robot's path slightly, specially where the channel is narrow. But, the success of the navigation system is not affected as shown in this example. The robot encounters no local minimum and navigates all the

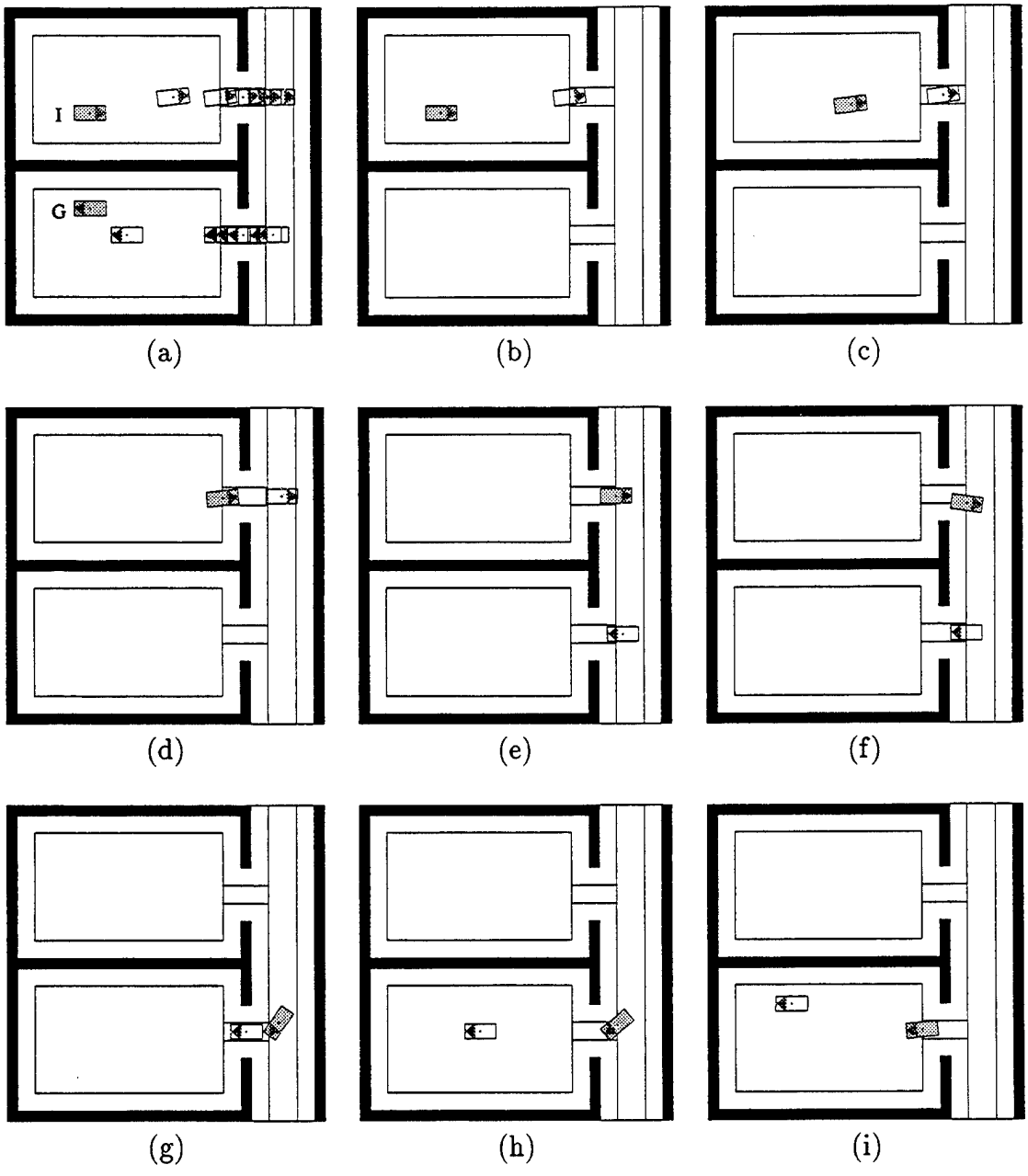
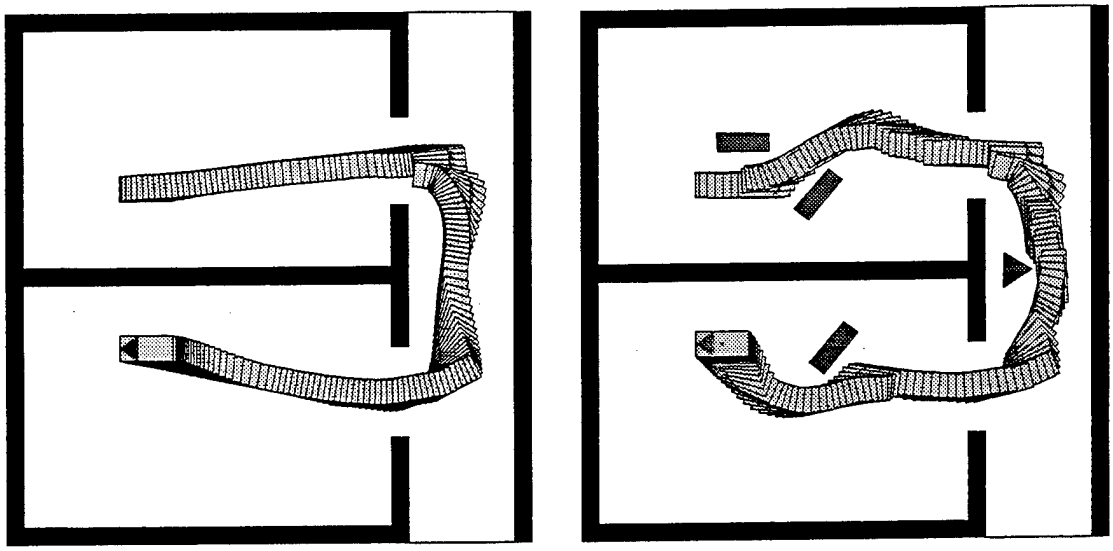


Figure 3.5: Shifting between intermediate goals in a channel



(a) No unexpected obstacle (b) With unexpected obstacles

Figure 3.6: Navigation in channel

way through the channel. Due to the damping term in the goal cell, the robot slows down near the goal configuration until it stops at the goal.

More examples in different environments are shown in Appendix A.

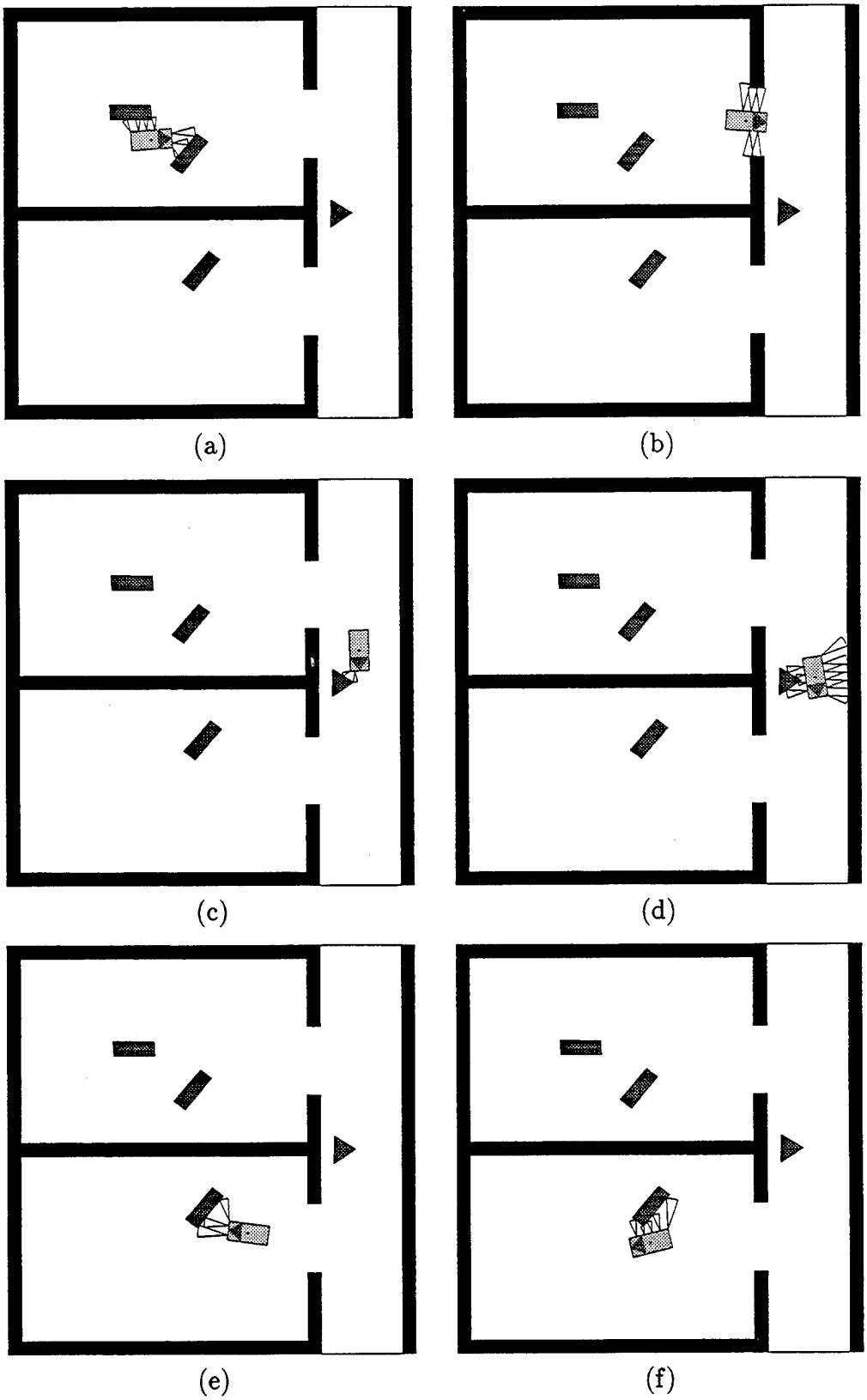


Figure 3.7: Detection of unexpected obstacles along a path

Chapter 4

Local Replanning

As long as the unexpected obstacles are small relative to the robot and sparsely scattered through the workspace, the potential fields defined in the previous chapter allow the robot to satisfactorily navigate in a channel. However, there may be local minima where the robot may get trapped. In particular, if there are big unexpected obstacles or unfortunate arrangements of small unexpected obstacles, local minima may have large attractive basins of attraction; the chances that the robot enters one of these basins increase.

Therefore, local minima should not be overlooked, and we must equip the navigation system with a procedure to escape them. Ideally, we would like the robot to avoid local minima by detecting them early enough, using the channel geometry information and the sensory data. However, it is not possible in general to recognize local minima until the robot has attained them. In this chapter we describe a procedure for detecting local minima and a method for escaping them.

4.1 Detection of Local Minima

We distinguish between two types of local minima in the potential field when there exist unexpected obstacles:

- A local minimum may appear in the neighborhood of the goal configuration, if this configuration is within the distance of influence of the unexpected-obstacle potential. In this case, the goal configuration is no longer a minimum of the potential. As mentioned in Section 3.2.2, such a local minimum can be avoided by selecting the distance of influence smaller than the distance between the goal configuration and the unexpected obstacles. In fact, the distance of influence of unexpected obstacles can be reduced toward 0 as the robot gets closer to the goal.

- A more common type of local minimum corresponds to a conflict between the channel and unexpected-obstacle potentials. It typically occurs in concavities made by arrangements of unexpected obstacles and the channel boundary. Figure 4.1 illustrates such a minimum: (a) shows a channel with three unexpected obstacles in the first cell; (b) shows the equipotential contours of the total potential, i.e., the combination of the channel potential and the unexpected-obstacle potential.[1] This second type of local minimum is far more difficult to deal with than the first type; it is the only type of local minimum that we will consider in this chapter.

In general, there is no simple way of detecting a local minimum in advance. Then, in theory, we can determine that the robot is trapped in a local minimum when the gradient of the potential vanishes. At best, we can provide methods for allowing the robot to escape the local minimum.

Mathematically, detecting a local minimum of the potential is a simple procedure. In practice, however, due to the discrete nature of robot's control and sensing, it is impossible for the robot to exactly land on a local minimum. Instead, the robot usually oscillates around the local minimum. Also, there are local minima whose attractive wells are so small that the robot may get away from them without any further help other than its own momentum. Local minima that cause a problem are

[1]The actual potential computed during navigation, however, would be slightly different, because, in Figure 4.1, the unexpected-obstacle potential was computed assuming complete knowledge of the unexpected obstacles.

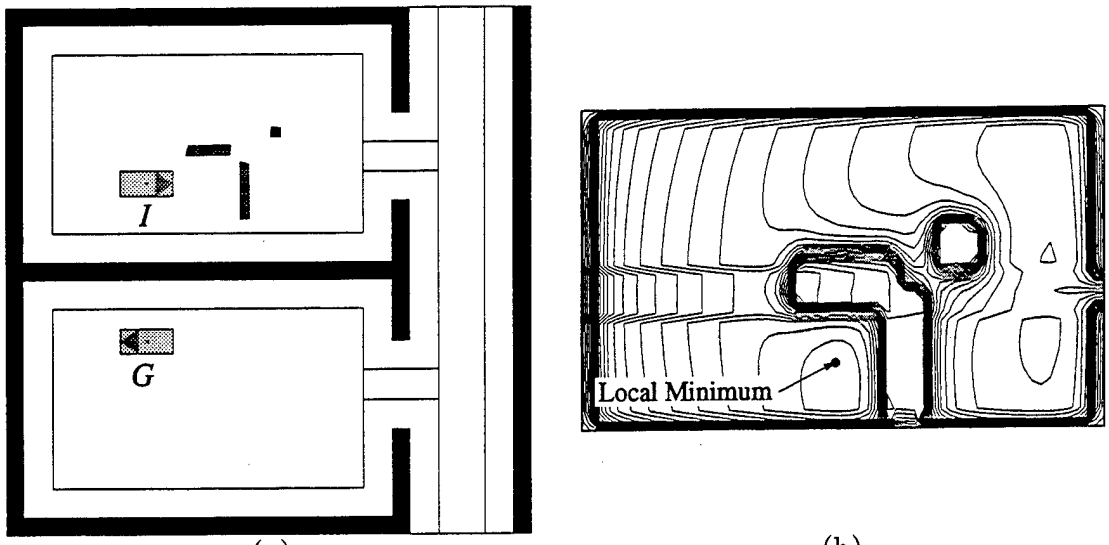


Figure 4.1: Typical local minimum

only the ones which have attractive wells large enough to trap the robot permanently. In the following, we describe a procedure that detects such local minima.

This procedure operates in two steps: the *alert* step and the *detect* step. When there is no local minima around the current robot's configuration, the robot's direction of motion deduced from the potential field usually does not change abruptly. Drastic changes are a strong indication that the robot has reached the bottom of the attraction well of a local minimum. We use such a drastic change in the direction of motion (e.g., a change greater than $\pi/2$) to set the "alert" flag. However, this is only a preliminary condition for deciding that the robot is in a local-minimum well. Indeed, narrow turns in the channel and intricate arrangements of unexpected obstacles may yield such changes in motion direction, without local minima. Moreover, such changes could also result from the fact that the unexpected-obstacle potential is time-varying, as the measurements of the sensors are updated at every instant. In order to make sure that the robot is actually stuck at a local minimum, we must wait for a while and check whether the robot continues moving. This is done by measuring the net distance to the configuration where the alert flag is set. If this distance does not

exceed a prespecified value after a prespecified waiting time,[2] the "detect" flag is set and the configuration of the robot at this instant becomes the local-minimum configuration q_{LM}.

The local-minima detection procedure significantly reduces incorrect detection of local minima. Although it does not completely eliminate incorrect diagnosis, experiments have shown that a more reliable detection of local-minimum configurations requires more time during which the robot tends to loop around. The penalty for incorrect local minimum detection is that we unnecessarily perform local replanning. But, as we will see below, the local replanning routine is usually very fast.

Figure 4.2 shows an example with two unexpected obstacles causing a local minimum: (a) shows the initial and goal configurations of the robot; (b) shows the path of the robot until it detects that it is trapped in a local-minimum configuration; (c) shows, in magnification, grid points corresponding to the part of the unexpected obstacles detected during this motion. We describe how to build such grid points using sensory data later in Section 4.3.2.

4.2 Escaping a Local Minimum

Once it has been determined that the robot is trapped in a local minimum, the robot needs a strategy to escape the basin of attraction of this minimum. If the robot's configuration space was 2-dimensional, a simple strategy would be to go around the unexpected obstacles [CZL89]. Only two possible directions of motion are possible, leaving the unexpected obstacle on the right or on the left of the robot, respectively. If the first direction fails, (e.g., by requiring the robot to move out of the channel), the second direction can be tried next. If both directions fail, this means that the unexpected obstacle obstructs the channel completely, and global replanning is needed.

However, this simple strategy is not applicable when the robot's configuration space is 3-dimensional (i.e., when the robot can both translate and rotate). Indeed,

[2]These values are determined experimentally considering the maximum speed of the robot and the rate of the closed-loop control in the reaction component of the navigation system.

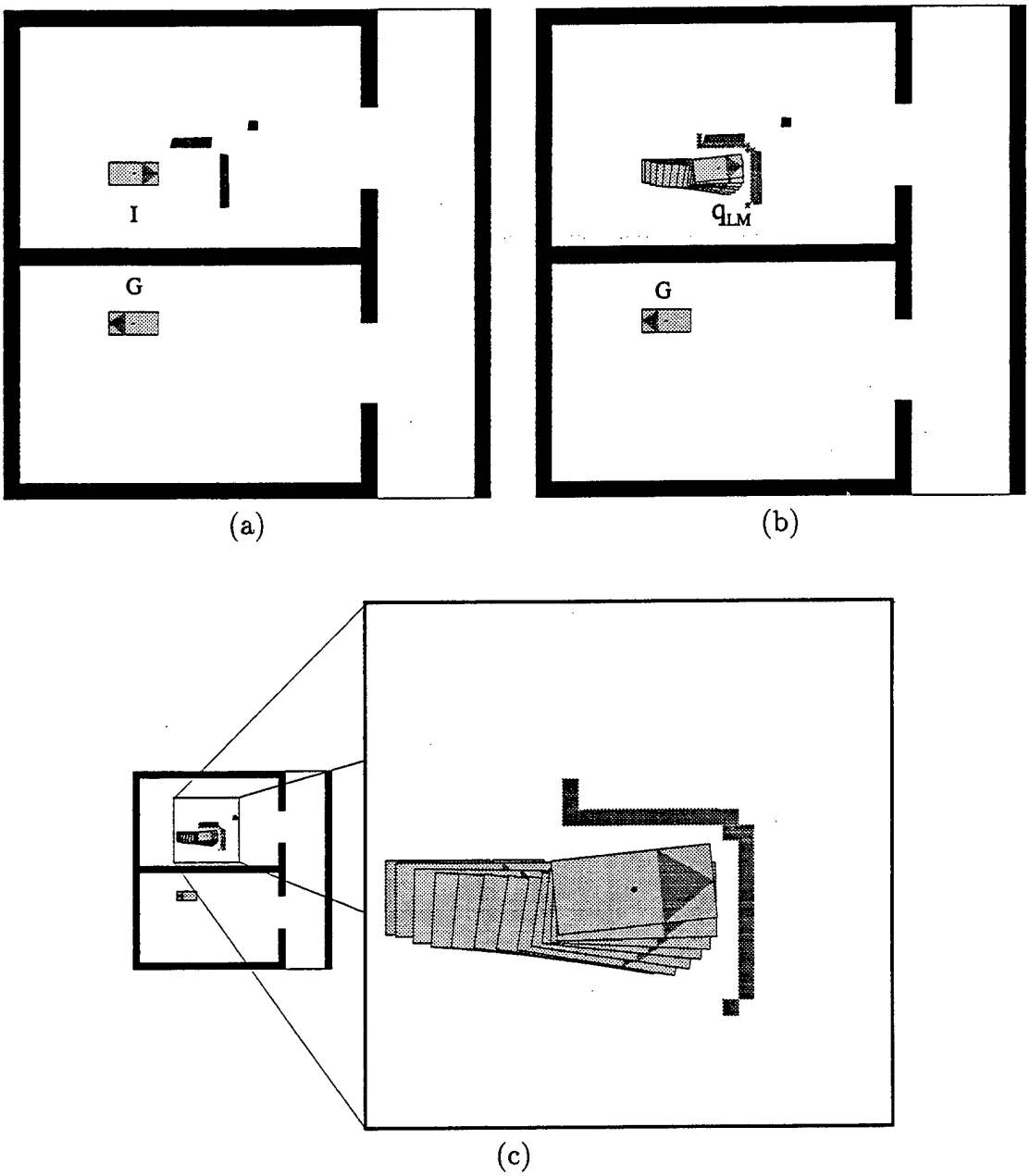


Figure 4.2: The robot gets trapped in a local minimum.

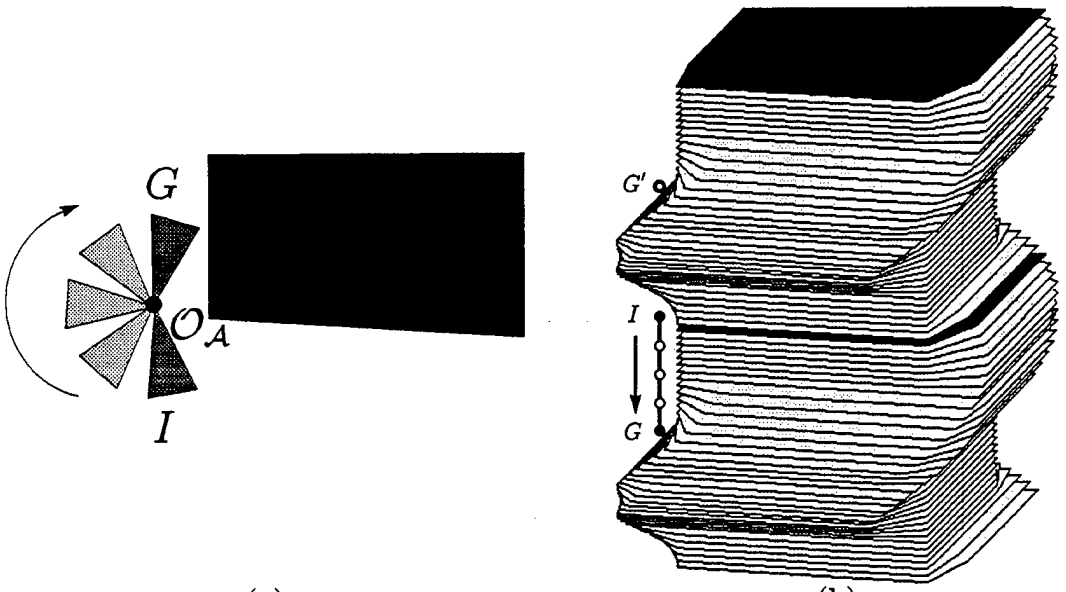


Figure 4.3: Local minimum in 3-dimensional configuration space

there are then an infinite number of directions that the robot can choose to move around the C-obstacle corresponding to the unexpected obstacle, and there is no way to select one that is always guaranteed to lead to a successful path. Besides, the orientation of the robot repeats itself every 2π along the orientation dimension (multiple connectedness of the configuration space), and this causes an additional difficulty in devising an escaping strategy. Figure 4.3 illustrates this difficulty with a simple example: (a) shows one possible path of a triangular robot from the initial configuration (I) to the goal configuration (G) in the 2-dimensional workspace. The path is achieved by rotating the robot clockwise around its reference point ($\mathcal{O}_A$). Figure (b) shows this path in the 3-dimensional configuration space as a collision-free line segment connecting I and G. The robot could have achieved the same goal orientation (shown G' in (b)) by rotating counterclockwise. Although the line segment connecting I to G' is shorter that the line segment $\overline{IG}$, i.e., $|\theta_G - \theta_I| > (2\pi - |\theta_G - \theta_I|)$ where θ_I and θ_G are the orientations of the robot at I and G, respectively, the segment $\overline{IG'}$ is not collision-free. Assume now the obstacle in Figure 4.3 is an unexpected one. With the potential described in Section 3.1.3, the navigation system, without knowing

in advance the existing obstacle, would take $G'(\theta_{G'} = \theta_G + 2\pi)$ as the attractive goal, which would make the robot rotate counterclockwise until it eventually reaches a local minimum caused by the unexpected obstacle.

Our navigation system escapes a local minimum by performing a local replanning operation, which we describe in the following two sections. The local replanning operation consists of two steps. First, a local path is planned to escape the local minimum by making use of the new information about the obstacles obtained through sensing. Second, this path is transformed into a valley-shaped potential that is then combined with the other potentials.

4.3 Local Path Planning

The path planning operation embedded in our local replanning method considers the cell of the channel where the local minimum has been detected. It searches this cell for a path connecting the local-minimum configuration[3] to the access gate of the next cell in the channel without colliding with the detected unexpected obstacles. If the minimum is close to the access gate of a cell, the path is planned in the union of two successive cells to provide the robot more space to move and escape the minimum. Various alternative path-planning techniques could be used to find a local path. However, many of these techniques either require the exact information about the geometry of the obstacles, or are too time-consuming. We make use of a very efficient potential-guided path planning technique.

The method uses a kind of Voronoi diagram, which represents the topology of the free space, and generates a local path that avoids as generously as possible both the detected unexpected obstacles and the boundary of the channel. This property is important because, in the next step, the local path is transformed into a valley-shaped potential and the robot travels in this new valley-shaped potential among the "yet to be detected" unexpected obstacles.

[3]More precisely, the configuration where the navigation system recognized that the robot was trapped in a local minimum.

4.3.1 Presentation of Local Path Planning Method

Let us first present the notion of a navigation function as described in [Lat91]: A navigation function is a potential function $\mathbf{U} : \mathcal{C}_{free} \mapsto \mathbf{R}$ with a minimum located at the goal q_{goal} whose domain of attraction includes the entire subset of the free space $\mathcal{C}_{free}$ that is connected to q_{goal}, except a finite set of isolated saddle points of $\mathbf{U}$. If a navigation function could be constructed, then a path generated by a depth-first search algorithm following the steepest descent of this function would be guaranteed to reach q_{goal}. Although it can be difficult to define an analytical navigation function over a space of arbitrary geometry, the computation of a numerical navigation function over a space represented in the form of a grid turns out to be much easier. A planning method based on such a numerical navigation function was first introduced in [BL91]. From here on, the numerical navigation function constructed in a grid is called a **grid potential**.

The grid-potential-based planning method consists of two steps: the generation of the grid potential and the search of a path using this grid potential. The planning can be performed in two different ways. The first method is to generate the grid potential directly in the configuration space of the robot and construct a path using the potential. The second method is to first generate a grid potential in the workspace of the robot and then to re-construct the grid potential in the configuration space during the search.

The operations in the first method consists of: transforming the detected obstacles into the corresponding configuration space $\mathcal{C}$-obstacles; discretizing the configuration space into a grid; computing a potential in the configuration space grid; tracking the steepest descent of the potential. The operations in the second method consists of: discretizing the workspace into a grid; computing a potential in the workspace grid; constructing a potential for the configuration space grid using the workspace potential and performing a best-first search using the re-constructed potential. In the second method, the configuration space potential may not be free of local minima. Nevertheless, as we show below, the total computational cost of the second method is usually much smaller than that of the first.

Except in the case where a robot moves with a fixed orientation, the number of the grid points in the configuration space is one order of magnitude larger than that in the workspace. For instance, in the case where a robot can both translate and rotate, the number of grid points is n^2 in the workspace, while it is of the order of n^3 in the configuration space,[4] where n is the number of grid points along the axes of the coordinate frame embedded in the workspace. The complexity of generating a navigation function is at least linearly proportional to the number of grid points. Hence, the cost of computing a navigation function in the configuration space (as is done in the first method) is much higher than in the workspace (as is done in the second method). In the second method, there is an additional cost for re-constructing the grid potential in the configuration space from the navigation function computed in the workspace. However, this re-construction needs to be done only for the grid points that are explored in the search process. In the unfortunate cases where the search procedure requires most of the grid points to be explored, the computational cost of the second method becomes equal to the cost of the first method, at worst. But such cases are rare. Besides, in the first method, there is the extra-cost of transforming detected obstacles into configuration space obstacles. This cost can be quite high, because the transformation operation requires generating algebraic representations of the detected unexpected obstacles from raw sensory data. The transformation is not needed in the second method. Therefore, in general, the second method is much faster than the first, and our navigation system uses this second method.

We apply this method with a rectangular **grid cell** $\mathcal{GK}$ placed across the **expanded cell**(s) considered for local path planning. Since these cells are usually rather small, $\mathcal{GK}$ is also small in most practical cases. The expanded cell is constructed by projecting a cell of the channel in the workspace of the robot and by enlarging it so that it includes all the points on the robot corresponding to the configurations of the robot in the cell. For computational simplicity we transform $\mathcal{GK}$ into a rectangle bounding the actual subset of the workspace swept out by the robot

[4]The number of grid points along the orientation axis (θ) is not necessarily the same as those along the translation axes (x and y), but it is proportional to them.

when its configuration varies over the channel cell. This slightly increases the size of the grid, but often by a negligible amount. Also, these additional grid points do not correspond to valid configurations of the robot in the channel; therefore, they are excluded during the search process.

$\mathcal{GK}$ is constructed by discretizing the axes of the frame $\mathcal{F}_{\mathcal{W}}$ embedded in $\mathcal{W}$. The free subset $\mathcal{GK}_{free}$ of $\mathcal{GK}$ is defined as:

$$\mathcal{GK}_{free} = \mathcal{GK} \setminus \bigcup_{i=1}^{n_{\mathcal{B}}} \boldsymbol{x}_{\mathcal{B}_i}$$

where $\boldsymbol{x}_{\mathcal{B}_i}$ denotes an **obstacle grid point**, a marked grid point indicating that it belongs to a detected obstacle, and $n_{\mathcal{B}}$ is the number of such points in $\mathcal{GK}$.

Before describing the computation of the grid potential in $\mathcal{GK}_{free}$, we explain, in the following subsection, how the sensory information about detected obstacles is used to mark points in $\mathcal{GK}$.

4.3.2 Local Map Building

While the robot is moving in the channel, the navigation system keeps track of the detected outline of the unexpected obstacles by mapping the sensory data into a grid obstacle model (see Figure 4.2 (b)). The grid model is associated with the grid cells $\mathcal{GK}_i$ $(i = 1, \ldots, p)$ of the channel, and it is continuously updated as the new sensory data become available. A function $\mathcal{M}_k$, associated with every sensor S_k, maps the sensed distance d_k (see Figure 3.3 (b)) and the current robot configuration to the corresponding point $\boldsymbol{x}_{\mathcal{B}}$ in $\mathcal{GK}$:

$$\mathcal{M}_k : (d_k, \boldsymbol{q}) \mapsto \boldsymbol{x}_{\mathcal{B}} \in \mathcal{GK}.$$

The path of the robot can be affected by the unexpected obstacles that are located not only in the current cell but also in the neighboring cells, and it may be important to keep a record of the unexpected obstacles even in the cells other than the current one. However, it is a waste of processing time to try mapping the sensory data

over all the cells in the channel at every instant, and it is often unnecessary unless the channel is small enough to be covered by sensory measurements at one location. In our implementation, the navigation system updates the grid obstacles only for a limited number of cells at a time. The number of the cells depends on the maximal range of sensing devices and the sizes of the neighboring cells.

Although we do not attempt to exploit the grid model to extract an algebraic representation for the detected unexpected obstacles, by increasing the resolution of the grid, we can make the model as precise as we wish. The fine resolution of the grid is desirable to capture the precise outline of the unexpected obstacles in the model, but the number of the grid points must be limited to save memory space and processing time. The resolution of the grid is limited by the resolution of the sensor (e.g., one half of the conic beam angle covered by a proximity sensor). It also depends on the speed of the robot (i.e., the size of a grid element does not need to be smaller than the distance traveled by the robot during one sensing/reaction loop).

4.3.3 Computation of Potential

The generation of the grid potential consists of computing workspace potentials (navigation functions) associated with *control points* in the robot and combining them into a configuration space potential.

Let a_i, $i = 1, \ldots, n_c$, be the control points selected in the robot $\mathcal{A}$. For a mobile robot moving in a plane in both translation and rotation, we typically take $n_c = 2$, since two points suffice to determine the robot's configuration. The control point a_i on $\mathcal{A}$ at configuration $\boldsymbol{q}$ is denoted by $a_i(\boldsymbol{q})$ in $\mathcal{W}$. With each point a_i we associate a navigation function $\mathbf{V}_i$ defined over $\mathcal{GK}_{free}$.

Once the workspace potentials have been computed over $\mathcal{GK}_{free}$, the configuration

space potential $\mathbf{U}$ at q can be computed by:[5]

$$\mathbf{U}(q) = \sum_{i=1}^{i=n_c} \alpha_i \mathbf{V}_i(a_i(q)).$$

where α_i is the (positive) weighting factor for each control point.

However, as discussed earlier, this potential function is not computed at every configuration, but only at the configurations where the value of $\mathbf{U}$ is needed for the search process. In order to compute $\mathbf{U}$ at a given configuration q, the position $a_i(q)$ of every control point is first computed and the value of $\mathbf{V}_i$ at the closest position of $a_i(q)$ in $\mathcal{GK}$ is used to compute $\mathbf{U}(q)$.

The computation of the workspace potential $\mathbf{V}_i$ is done in three steps. First, a subset $\mathcal{S}$ of $\mathcal{GK}_{free}$, called a **skeleton**, is extracted. Second, $\mathbf{V}_i$ is computed in $\mathcal{S}$. Third, $\mathbf{V}_i$ is computed in the rest of $\mathcal{GK}_{free}$. The three steps are described in detail in [Lat91]. In the following, We present them with modifications and illustrations that are specific to our problem.

In order to extract $\mathcal{S}$, the L^1 (Manhattan) distance $d_1(x)$ from every point $x \in \mathcal{GK}_{free}$ to the grid points on the boundary (**boundary point**) of $\mathcal{GK}_{free}$ is computed using a wavefront propagation algorithm. L^1 is chosen for computational simplicity, i.e., the integer arithmetic throughout the computation of the potential, but other distance metrics, e.g., L^2 (Euclidean distance), could be used instead. The differences in the resulting paths are relatively minor, in general. Besides, the effect of choosing a particular distance metric on navigation is even less significant here since the resulting path will not be strictly followed by the robot, but rather used as a guide.

The wavefront propagation starts from the boundary points of $\mathcal{GK}_{free}$. First, these points are identified, and the value of d_1 at these points are set to 0. Then, the d_1 value is increased by 1 and assigned to every neighbor of these points which does

[5]There are many variations in combining the workspace potentials to compute the configuration space potential, and each of them gives a different "flavor" to the resulting path. However, we use the simple weighted addition because no single method seems to be always superior to another, in general. See [Lat91] for other combinations.

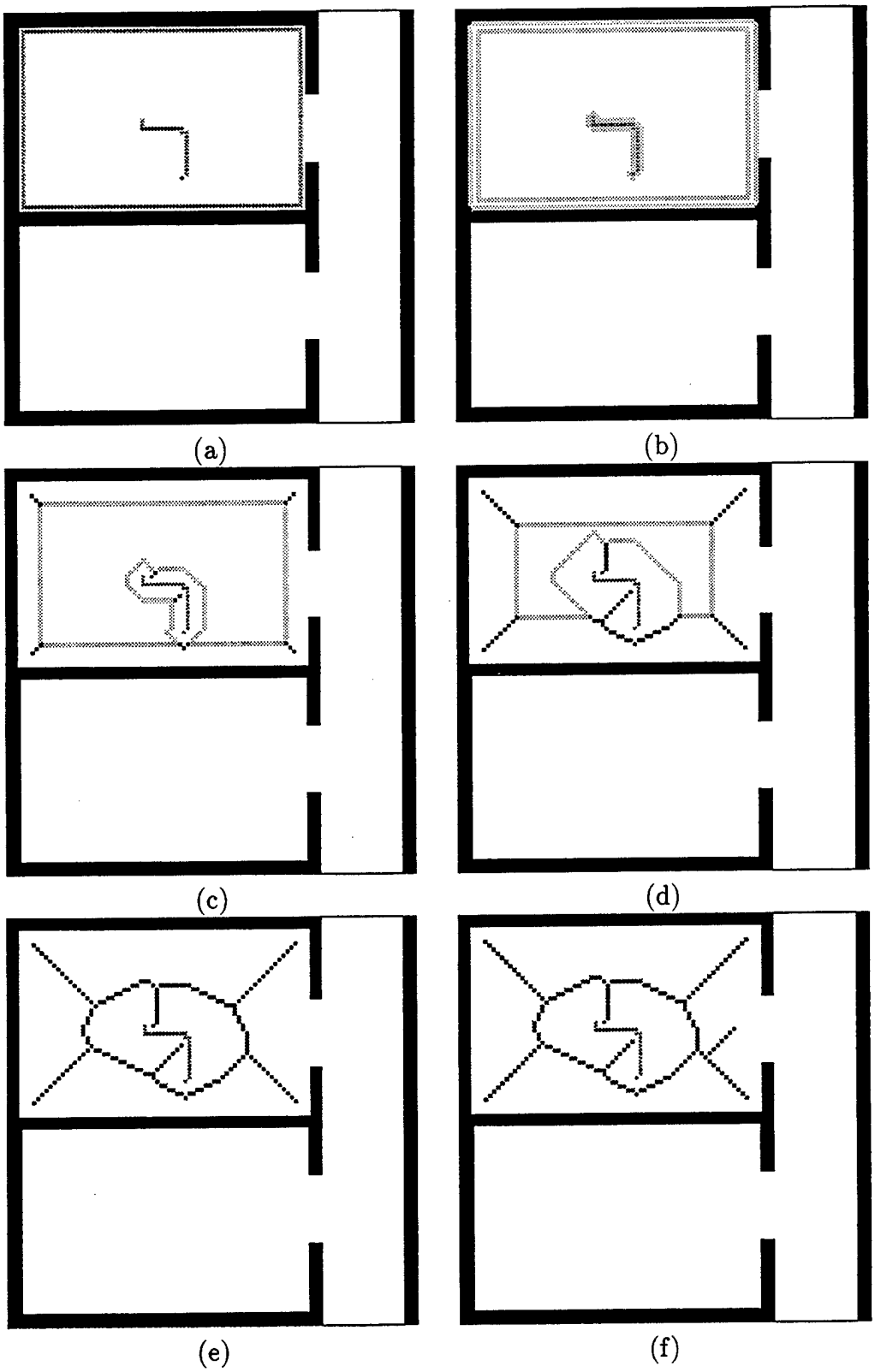


Figure 4.4: Skeleton constructed by wavefront propagation

not have a value of d_1 assigned to it yet. This propagation of the d_1 value continues until every grid point in $\mathcal{GK}_{free}$ is explored. The boundary consists of the bounding rectangle of $\mathcal{GK}$ and the boundary of the regions formed by the marked point $\boldsymbol{x}_{\mathcal{B}}$.

In parallel to the propagation, we construct $\mathcal{S}$ as the set of points where the wavefronts issued from the boundary points meet. This is done by propagating not only the values of d_1, but also the boundary points that are at the origin of the propagation. The construction of $\mathcal{S}$ is completed by connecting the grid point $\boldsymbol{x}_{goal}$ closest to $a_i(\boldsymbol{q}_{goal})$ to $\mathcal{S}$. This connection consists of a path σ following the steepest ascent of the d_1 map in $\mathcal{GK}_{free}$. This path is appended to $\mathcal{S}$. Figure 4.4 illustrates the wavefront propagation in the example workspace shown in Figure 4.2: (a) shows the boundary of $\mathcal{GK}_{free}$; (b)-(d) show successive snapshots during the propagation (the current wavefront is shown in gray, while the parts of the skeleton already computed are shown in black.) ; (e) shows the skeleton constructed at the end of the propagation; (f) shows the final skeleton including the path σ from the goal a_{goal} at the cell exit to the skeleton.

Next, $\mathbf{V}_i$ is computed in $\mathcal{S}$ by another wavefront propagation algorithm restricted to $\mathcal{S}$ and starting from the grid point $\boldsymbol{x}_{goal}$. The algorithm uses the d_1 map previously computed in order to guide the propagation. At first, the starting grid point $\boldsymbol{x}_{goal}$ is given the value 0 of $\mathbf{V}_i$, and it is inserted in a list sorted by decreasing value of d_1. The grid point $\boldsymbol{x}$ at the top of the list (i.e., the one having the largest value of d_1, the furthest away from the boundary of $\mathcal{GK}_{free}$) is removed from the list and used at the next step of the propagation. Every neighbor of $\boldsymbol{x}$ in $\mathcal{S}$ whose potential has not been computed yet receives a potential value of $\mathbf{V}_i(\boldsymbol{x}) + 1$, and is inserted in the list. This procedure continues until the list becomes empty, i.e., when all the points in $\mathcal{S}$ accessible from $\boldsymbol{x}_{goal}$ has been assigned a potential value. Figure 4.5 illustrates this computation. The skeleton elements shown black are those which have been attained at the current stage of the propagation. The parts of the skeleton located in the widely open areas are explored earlier, shown in (a) through (c), than the ones in the narrow areas. This results in a discontinuity in the value of skeleton potential when two wavefronts meet in a closed loop of the skeleton (for example, see the lower part

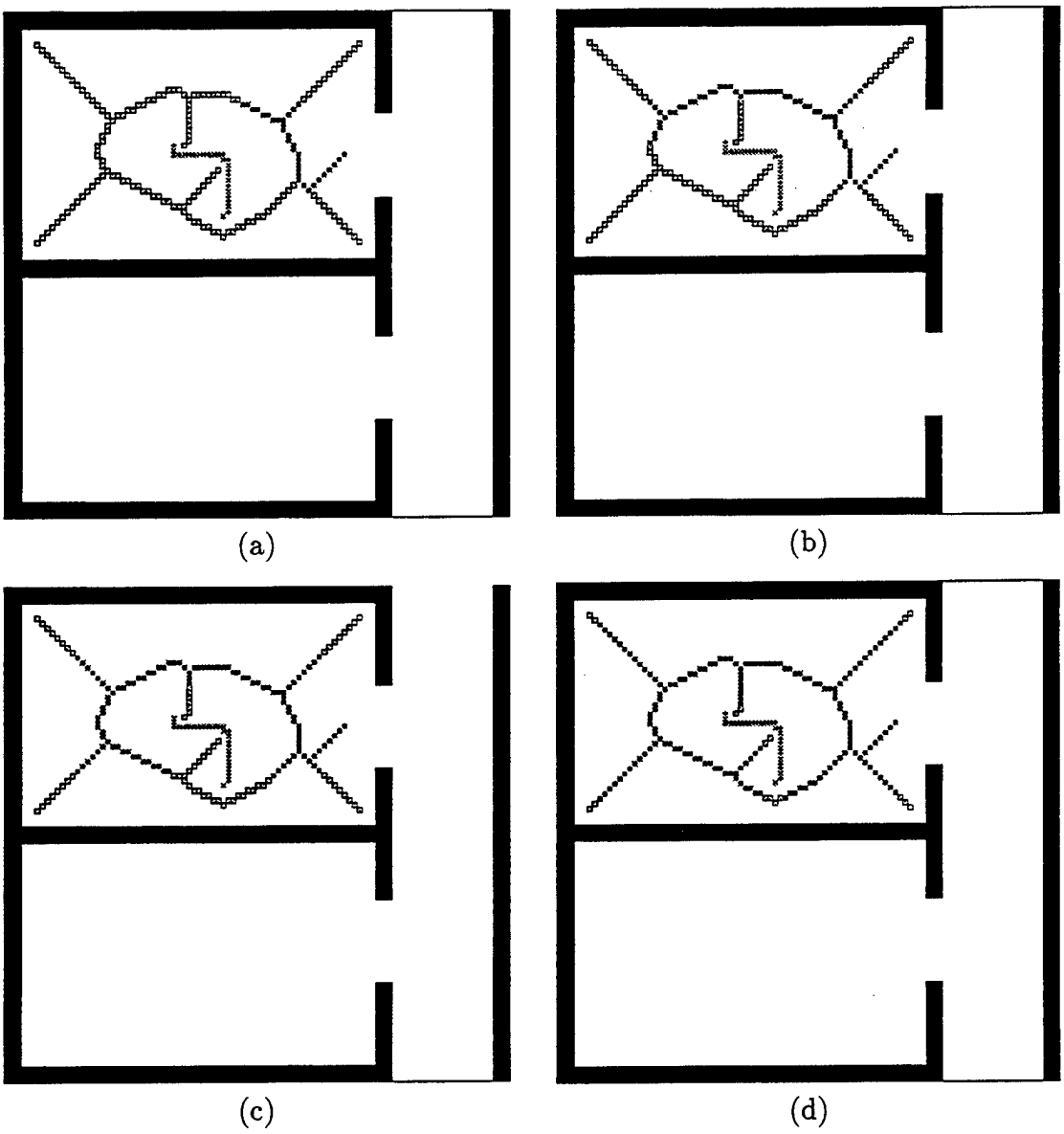


Figure 4.5: Computation of potential in the skeleton by wavefront propagation

of the skeleton in Figure (d)). This discontinuity results in a "cliff" in the final grid potential (see Figure 4.7) with two opposite gradients on each side of the cliff. So, in the search process, the path is created such that it goes through an most widely open area rather than through the narrow area.

Finally, the potential $\mathbf{V}_i$ in the rest of $\mathcal{GK}_{free}$ is computed, again, by a wavefront propagation starting from every point in $\mathcal{S}$ with its initial potential value assigned in the previous step. Each neighbor $\boldsymbol{x}'$ of every grid point $\boldsymbol{x} \in \mathcal{S}$ is given a potential value of $\mathbf{V}_i(\boldsymbol{x}) + 1$. The propagation continues similarly and terminates when all the accessible grid points from $\boldsymbol{x}_{goal}$ have been explored. Figure 4.6 illustrates this final propagation process. Figures (a) through (d) show snapshots during the propagation. The current wavefronts are shown in gray, and the grid points inside these wavefronts are those whose potential have been computed already. Figure 4.7 shows the 3-dimensional perspective view (a) and the equipotential contours (b) of $\mathbf{V}_i$ for $\mathcal{GK}$. The potential at the obstacle grid points is not determined; arbitrary high values are given to these points for the search process. The resulting potential has no other local minimum than the intermediate goal at the exit gate of the cell.

4.3.4 Grid Search

The Best First Search technique [Lat91] is used to compute a local path γ connecting $\boldsymbol{q}_{LM}$ and $\boldsymbol{q}_{goal}$ using the configuration space potential $\mathbf{U}$ defined as above. However, because it is computed using a few control points only, this potential does not guarantee that there will be no overlap between the robot and the boundary of $\mathcal{GK}_{free}$ during the search. Therefore, at each step of the search, all the points $\mathcal{A}(\boldsymbol{q})$ of the robot at the current configuration $\boldsymbol{q}$ must be checked to lie in $\mathcal{GK}_{free}$. Collision-checking is performed using the grid outline of the robot (with the same resolution as the grid cell). Such collision-checking technique is justified by the assumption that motion increments in the search are small enough so that if the robot is in free space near the outline of an obstacle at one step, it cannot "jump" over the outline and be in free space at the next step.

Figure 4.8 shows a path generated by the Best First Search method: (a) shows

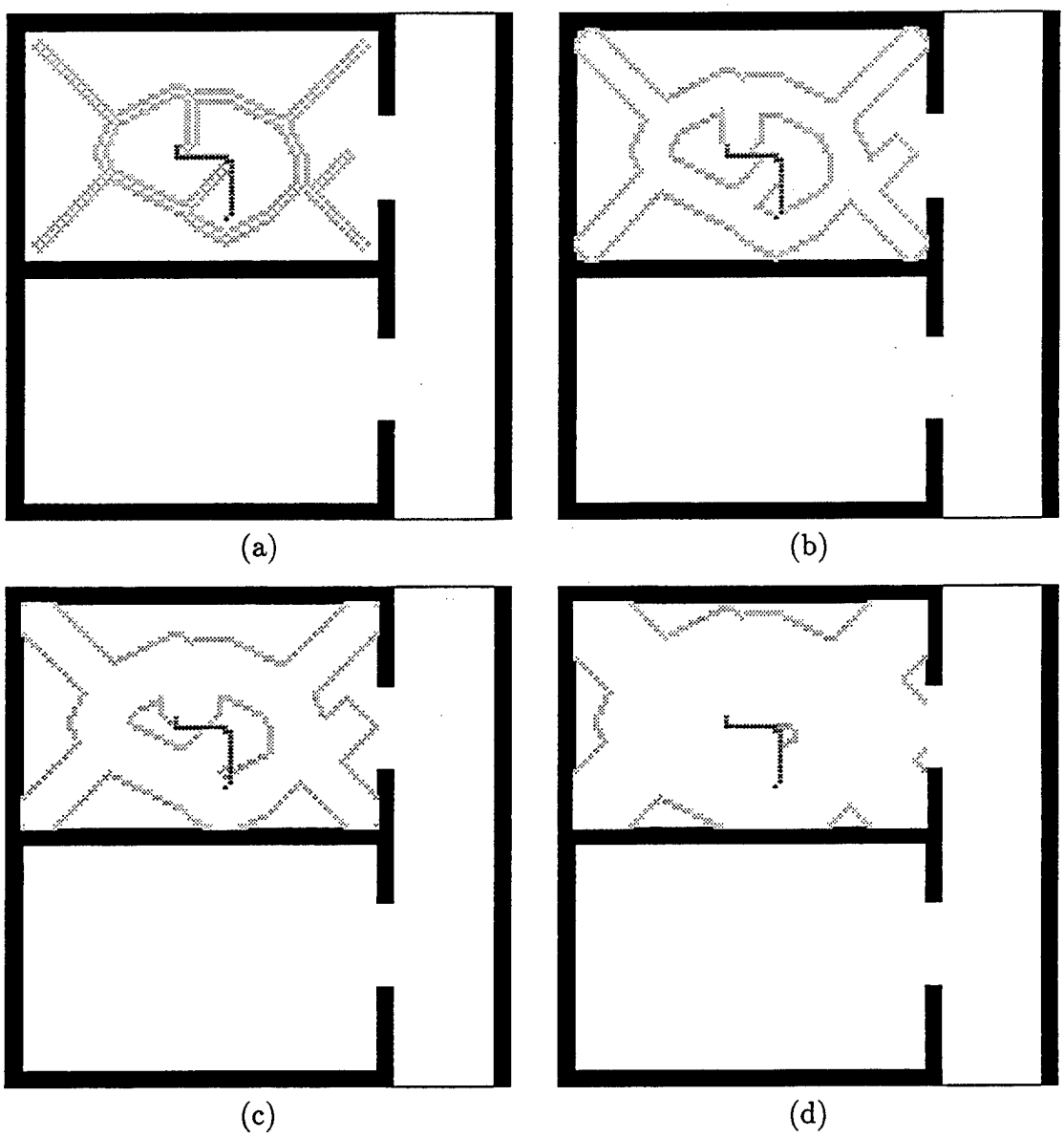
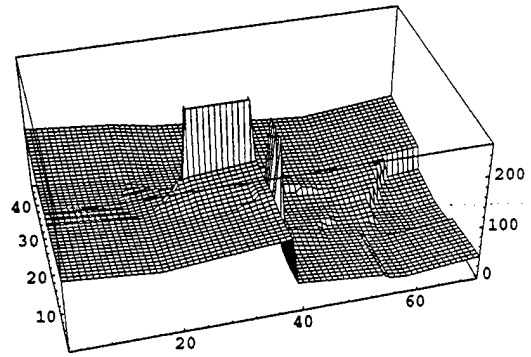
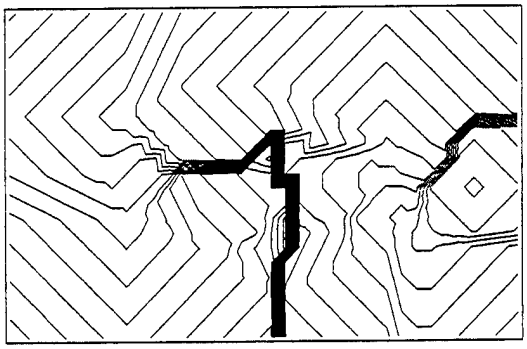


Figure 4.6: Potential propagation from the skeleton

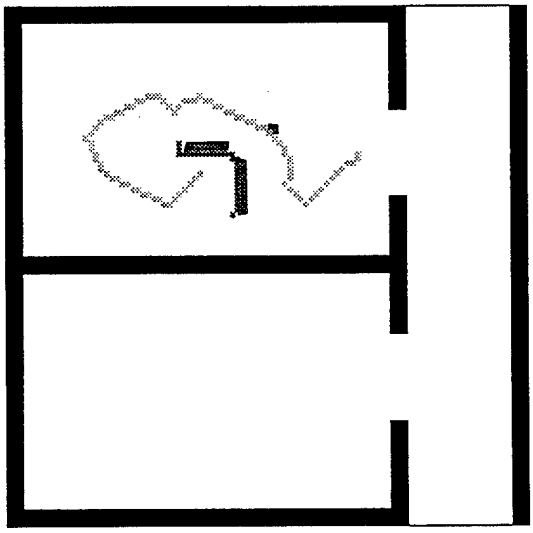


(a) 3-dimensional perspective view

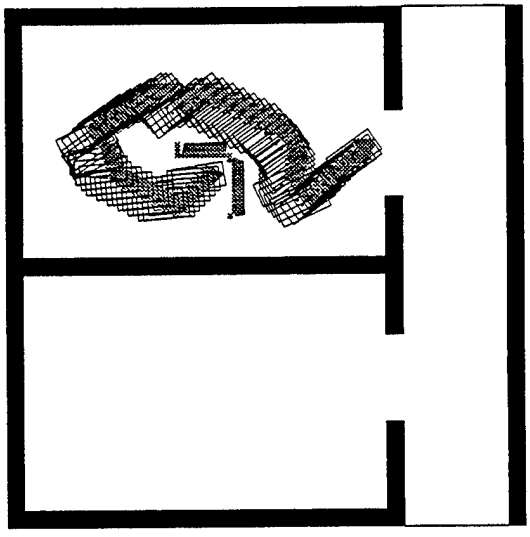


(b) equipotential contours

Figure 4.7: Grid potential in a cell



(a) grid nodes along the local path



(b) configurations at the nodes

Figure 4.8: Local path

the grid points corresponding to the reference point of the robot along the path; (b) shows (with transparent robot) the configurations of the robot at all the grid points along the path. Note that the local path leads the robot to collide with a still undetected obstacle. This potential collision stresses the need for transforming the generated local path into a less committed motion plan (valley-shaped potential). We now describe this final transformation.

4.4 Valley-shaped Potential

4.4.1 Principle

Once the path planner has succeeded generating a local path γ, the reaction component must drive the robot along γ to escape the local minimum. It must also avoid collision with other (still undetected) unexpected obstacles and stay inside the channel. Following the local path γ exactly may lead the robot to collide with the new undetected obstacles (or the still unsensed parts of previously detected obstacles), as illustrated in Figure 4.8 (b). This leads us to transform the local path into a potential function to be combined with the channel potential and the unexpected-obstacle potential.

We relax the commitment to the local path γ by transforming it into a valley-shaped potential field $\mathbf{U}_\gamma$ (with γ as the "bottom" of the valley) defined over the one or two cells containing γ. We next add $\mathbf{U}_\gamma$ to $\mathbf{U}_c$ and $\mathbf{U}_s$, and make the robot follow the negated gradient of the total potential:

$$\mathbf{U}(\boldsymbol{q}) = \mathbf{U}_c(\boldsymbol{q}) + \mathbf{U}_s(\boldsymbol{q}) + \mathbf{U}_\gamma(\boldsymbol{q}).$$

$\mathbf{U}_\gamma$ is constructed in such a way that at any $\boldsymbol{q} \in \gamma$, the vector $-\nabla\mathbf{U}_\gamma$ points along the tangent of γ, and at any $\boldsymbol{q} \notin \gamma$, $-\nabla\mathbf{U}_\gamma$ is the sum of two components, one pointing along the tangent of γ, the other pointing toward γ (see Figure 4.9). Hence, the robot is attracted back to γ whenever it deviates from it. It is also pulled toward the end extremity of γ. If another local minimum is detected, it is treated in the

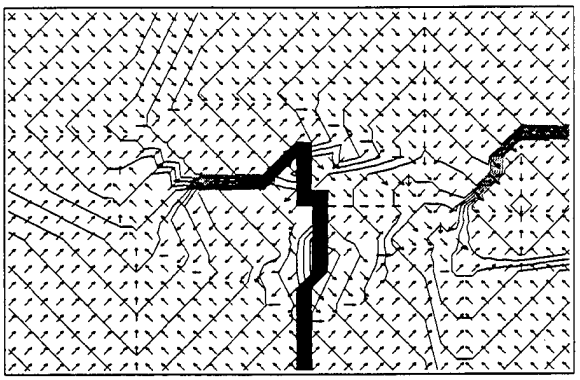


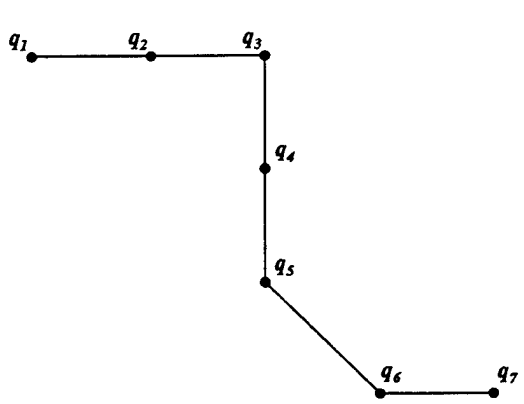
Figure 4.9: Vector field induced by the valley potential

same way as the previous one, i.e., by generating another local path (the previous local path is forgotten). When the robot reaches the end of γ, the potential field is re-established to $\mathbf{U}_c + \mathbf{U}_s$. In the following subsection, we describe in detail how the valley potential is computed from a locally planned path.

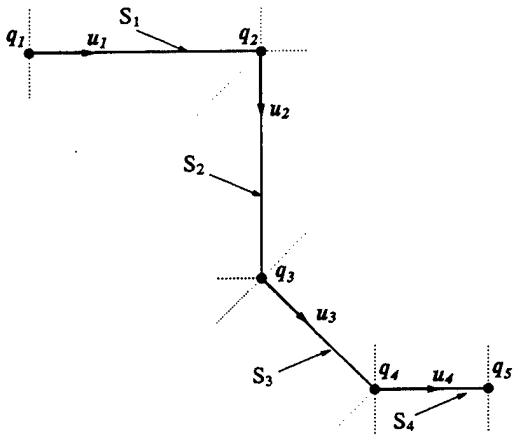
4.4.2 Implementation

The local path lies in the configuration space grid. Hence, it is made of line segments connecting a sequence of configurations in the grid cell, starting at the local minimum and ending at the intermediate goal of the cell exit. Figure 4.10 (a) illustrates the 2-dimensional projection of a portion of a typical local path. The local path γ may include two or more successive segments that are collinear; then, these segments are merged into a single one to reduce the number of segments as shown in Figure 4.10 (b). The sequence of obtained segments is called a **spine**. The vertices of the spine are denoted by $\boldsymbol{q}_i$ $(i = 1, \ldots, n_\gamma)$. A segment of the spine between two vertices $\boldsymbol{q}_i$ and $\boldsymbol{q}_{i+1}$ is denoted by s_i. The valley potential $\mathbf{U}_\gamma$ is the union of several potentials $\mathbf{U}_{s_i}$. Each $\mathbf{U}_{s_i}$ is defined over the influence region of a segment s_i $(i = 1, \ldots, n_\gamma - 1)$:

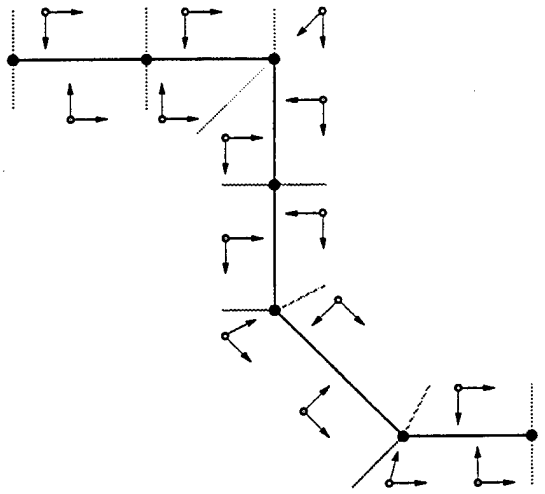
$$\mathbf{U}_\gamma(\boldsymbol{q}) = \mathbf{U}_{s_1}(\boldsymbol{q}) \cup \mathbf{U}_{s_2}(\boldsymbol{q}) \cup \ldots \cup \mathbf{U}_{s_{n_\gamma - 1}}(\boldsymbol{q}).$$



(a) local path



(b) merged local path



(c) valley potential defined on each segments

Figure 4.10: Spine of a valley projected on a plane

The influence region $\mathcal{R}_{s_i}$ of a segment s_i is a set of configurations defined by:

$$\mathcal{R}_{s_i} = \{q \mid d(q, s_i) = \min_{j=1}^{n_\gamma - 1} d(q, s_j)\}$$

where $d(q, s_j)$ is the shortest distance from q to s_j. Configurations that achieve minimal distance with more than one segment are placed in the influence region of one of these segments selected as described below. Hence, at every configuration q, $\mathbf{U}_\gamma(q)$ is completely determined by a single segment s_i. Therefore, instead of computing the complete valley potential in advance, the potential is computed for each configuration as the robot moves along the successive segments of the spine. When the configuration q of the robot enters the influence region of a segment s_i, the robot becomes under the influence of $\mathbf{U}_{s_i}$. We call this segment the **effective** segment. The computation of $\mathbf{U}_{s_i}$ is performed in two steps: (1) find the effective segment for q; (2) compute the two components, $\mathbf{U}_t$ and $\mathbf{U}_n$, of $\mathbf{U}_{s_i}$. As described in the previous section, $\mathbf{U}_t$ is used for moving the robot along the tangent of s_i, and $\mathbf{U}_n$ for attracting the robot back to s_i (see Figure 4.10 (c)).

Instead of computing the influence region of s_i in advance, we compute, at every cycle in the reaction control loop, the shortest distances from the current configuration to the current effective segment (in the beginning, the first segment is considered effective) and every following segments in the spine. A segment closest to q is selected as the effective segment for the current configuration. When the configuration is at the same distance to two or more segments, the furthest segment in the sequence is selected as the effective segment, thus giving a higher priority to the forward motion along the spine. Excluding the segments that are behind the current effective segment prevents the robot from oscillating between two segments, especially when two segments are close to each other and have an acute angle between them. It also reduces computation.

Computing the shortest distance of a point to a line segment is simple. However, if there are many segments in the spine, the cost of computation may not be ignored, because the computation is repeated at every cycle of the reactive control loop. We

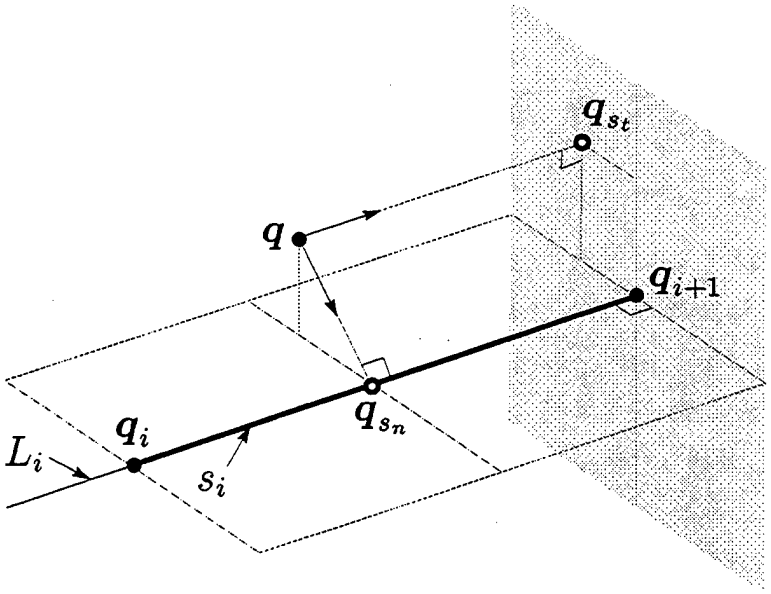


Figure 4.11: Effective segment and its shadow points

reduce the computation by comparing the bounding box that contains the current effective segment and the current configuration to the bounding boxes of the remaining segments and precluding the segments that are far away.

Figure 4.11 shows a configuration of the robot q and its effective segment s_i. Let L_i be the supporting line of s_i. Let the **normal shadow** point of q be the point $q_{s_n} \in L_i$ such that the line segment $\overline{qq_{s_n}}$ is normal to L_i. Let the **tangential shadow** point be the point q_{s_t} in the plane perpendicular to L_i at q_{i+1} such that the line segment $\overline{qq_{s_t}}$ is parallel to L_i. If q_{s_n} lies outside of s_i, q_i becomes q_{s_n}. Given q_{s_t} and q_{s_n} of s_i, we compute $\mathbf{U}_{s_i}$ as follows:

$$\mathbf{U}_{s_i} = \mathbf{U}_t + \mathbf{U}_n$$

$$\mathbf{U}_t = \frac{1}{2} K_t \rho_t^2(q)$$

$$\mathbf{U}_n = \frac{1}{2} K_n \rho_n^2(q)$$

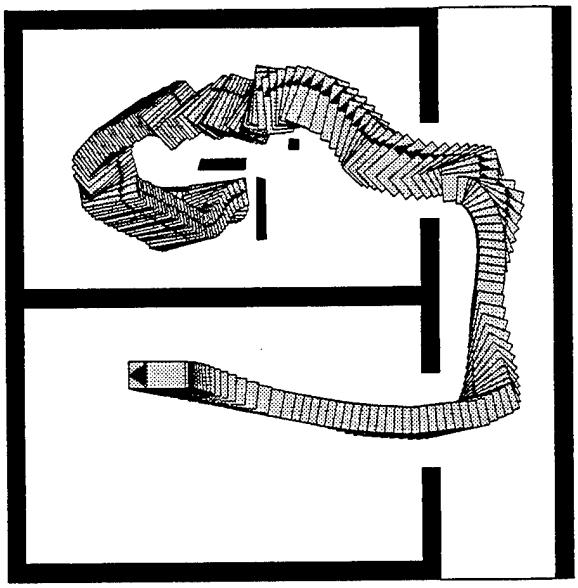


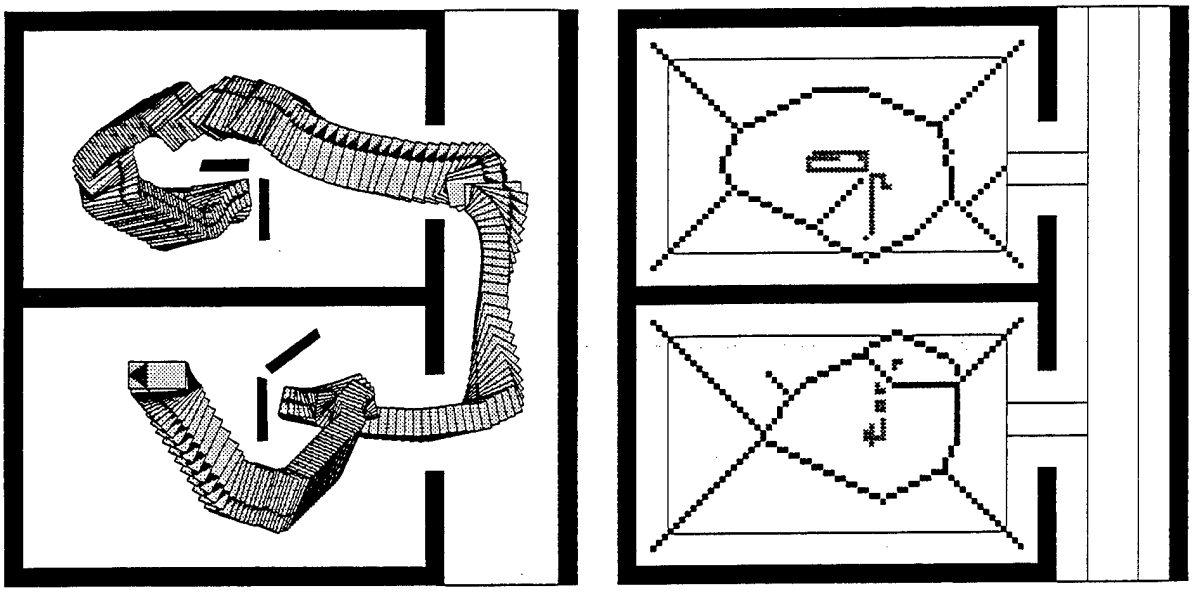
Figure 4.12: Escaping a local minimum

where

- K_t and K_n are scaling factors,
- $\rho_t(\boldsymbol{q})$ is the distance between $\boldsymbol{q}$ and $\boldsymbol{q}_{s_t}$ (i.e., $\rho_t(\boldsymbol{q}) = d(\boldsymbol{q}, \boldsymbol{q}_{s_t})$),
- $\rho_n(\boldsymbol{q})$ is the distance between $\boldsymbol{q}$ and $\boldsymbol{q}_{s_n}$ (i.e., $\rho_n(\boldsymbol{q}) = d(\boldsymbol{q}, \boldsymbol{q}_{s_n})$).

4.5 Examples

Figure 4.12 shows the path followed by the robot to escape the local minimum encountered in Figure 4.2. Due to the combination of $\mathbf{U}_c$, $\mathbf{U}_s$ and $\mathbf{U}_\gamma$, this path slightly differs from γ and is free of collision with any obstacle. Figure 4.13 shows another example where the robot encountered two local minima during navigation. (a) shows the path followed by the robot; (b) shows two skeletons generated by local replanning, performed twice during the navigation. The skeleton in the first cell was generated after the first local minimum had been attained, and the second skeleton



(b) the complete path (a) the skeletons

Figure 4.13: Navigation of the robot with two local minima

in the last cell was generated after the second local minimum had been attained. In both the first and last cell, a grid of size of $69 \times 44 \times 36$ is used for local path planning.

Chapter 5

Global Replanning

In the previous chapter we discussed local replanning to escape local minima. In most cases, the presented method successfully generates an escape route for the robot. However, for some arrangements of unexpected obstacles, our local replanning method fails. The reason for such failure is that replanning is restricted to one or two channel cells. In this chapter we extend the local replanning method to multiple cells. We also present a global replanning method which generates an alternative channel when it is detected that the current channel is completely obstructed. Throughout this chapter, we simplify our presentation by assuming, without loss of generality, that the robot is circular, which yields a 2-dimensional configuration space. In particular, this assumption results in simpler figures.

5.1 Local Replanning Failures

The single-cell (or double-cell) local replanning may fail to find a path for one of the following two reasons: (1) The current channel is completely obstructed; (2) The local minimum cannot be escaped by staying in the current cell. Let us illustrate each case with examples.

Figure 5.1 illustrates an example where our local replanning fails because the channel is completely obstructed by two unexpected obstacles (shown in dark gray). First, in (a), the robot falls into a local minimum due to the unexpected obstacles.

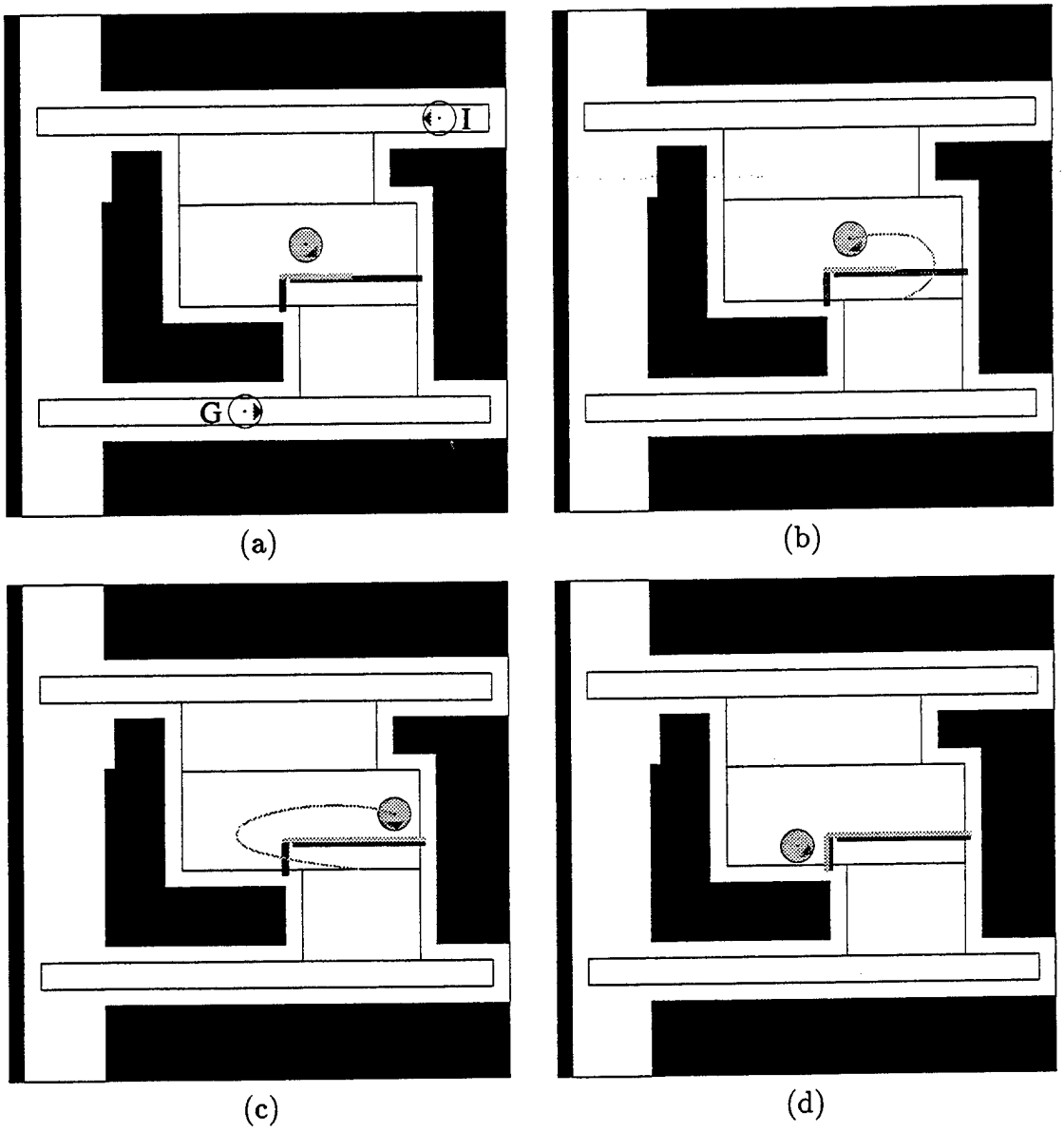


Figure 5.1: Local replanning failure due to channel obstruction

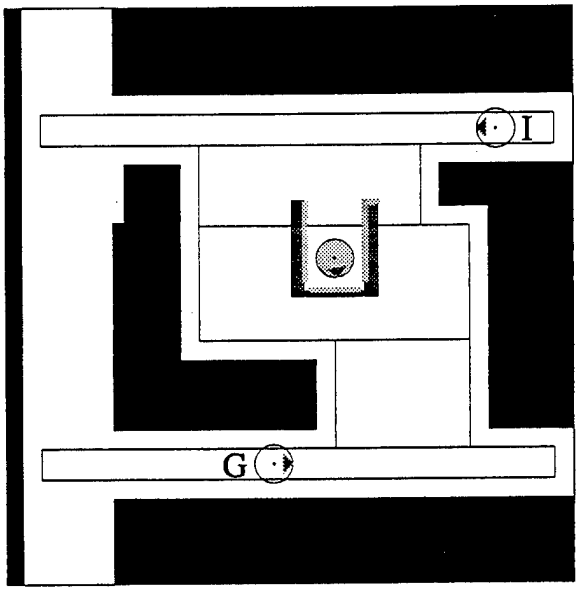


Figure 5.2: Local replanning failure due to locality of replanning

However, when this minimum is attained, only a subset of the unexpected obstacles has been detected (shown in light gray). Therefore, there is no way for the navigation system to immediately recognize that the channel is completely obstructed. The robot escapes the local minimum as any other local minimum using the method described in the previous chapter. Then, it falls into a second local minimum that it tries to escape again. This escape process is repeated until enough sensory information about the unexpected obstacles has been accumulated so that local replanning fails to find a path. Figures (b) through (d) show the sequence of the escape trials with each locally planned path displayed as a gray line.

Figure 5.2 illustrates the second case of failure of our replanning method. Here local replanning fails because it is restricted to the current cell. Three unexpected obstacles create a dead-end lying over two cells. This dead-end results in a local minimum where the robot gets trapped. Although there still exists an escape path in the current channel, the path planner used by the local replanning method only considers the cell of the local minimum, and it fails to find a path in this cell.

In the above examples, paths to the goal G do exist despite the local replanning

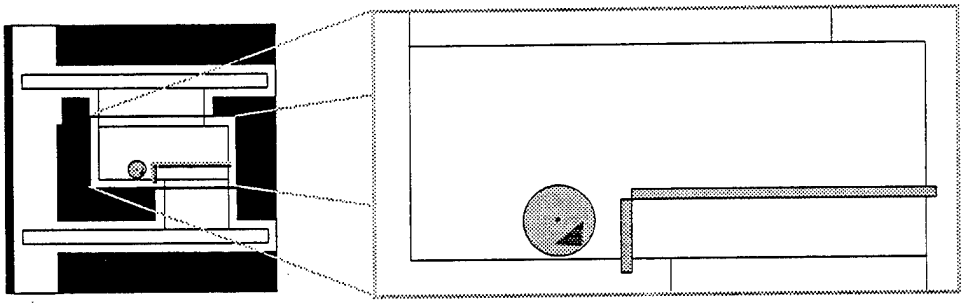
failures. The first case is solved by global replanning, i.e., finding an alternative channel, using the additional information provided by the sensors. The second case can be also solved by alternative channel generation, although it can be solved by multi-cell local replanning, which we will describe later. First, we describe alternative channel generation.

5.2 Alternative Channel Generation

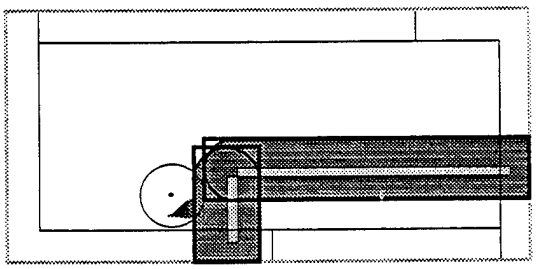
An alternative channel is generated by the planning component of the navigation system. The operation, as described in Chapter 2, consists of two steps: updating the connectivity graph and searching the graph for a new channel.

Some empty cells in the current connectivity graph become mixed cells due to the detected unexpected obstacles represented by obstacle grid points. Mixed cells are identified by merging adjacent obstacle grid points into rectangular obstacles and by computing the intersection between empty cells in the current connectivity graph and rectangles bounding C-obstacles corresponding to these new obstacles. Once the connectivity graph is updated, it is searched for a new sequence of the empty cells between the cell containing the local-minimum configuration, where local replanning has failed, and the cell containing the goal configuration. When a new channel is found, navigation resumes in this channel. If an alternative channel cannot be found, the planning component reports failure and the navigation system stops.

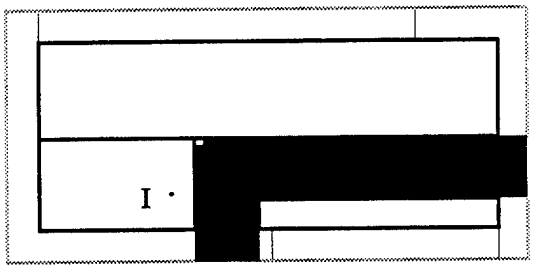
Figure 5.3 illustrates alternative channel generation for the example of Figure 5.1. Figures (a), (b), and (c) illustrates, in magnification, three stages of cell decomposition: (a) shows the local-minimum configuration in the current cell with obstacle grid points transformed into two rectangular obstacles (in gray); (b) displays the rectangles (in thick lines) bounding the C-obstacles (in dark gray) corresponding to the two new rectangular obstacles; (c) shows the decomposed empty cells (in thick lines) in the current cell. Figure (d) depicts the new channel connecting the new initial configuration I and the goal configuration G. Figure 5.4 shows the new channel generated for the example of Figure 5.2 using the same global replanning method. In this case,



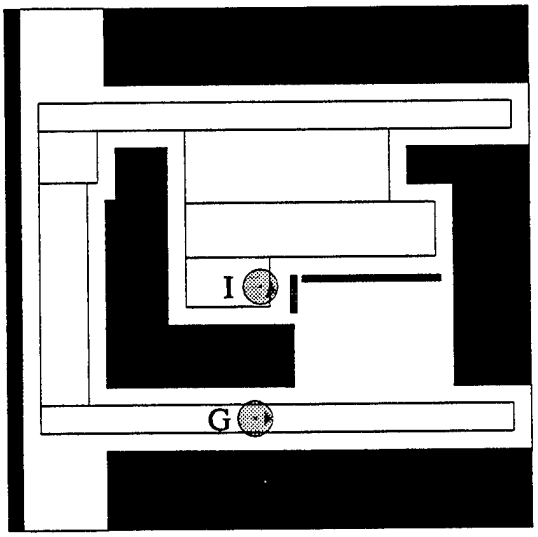
(a) rectangular obstacle models



(b) bounding rectangles of $\mathcal{C}$-obstacles

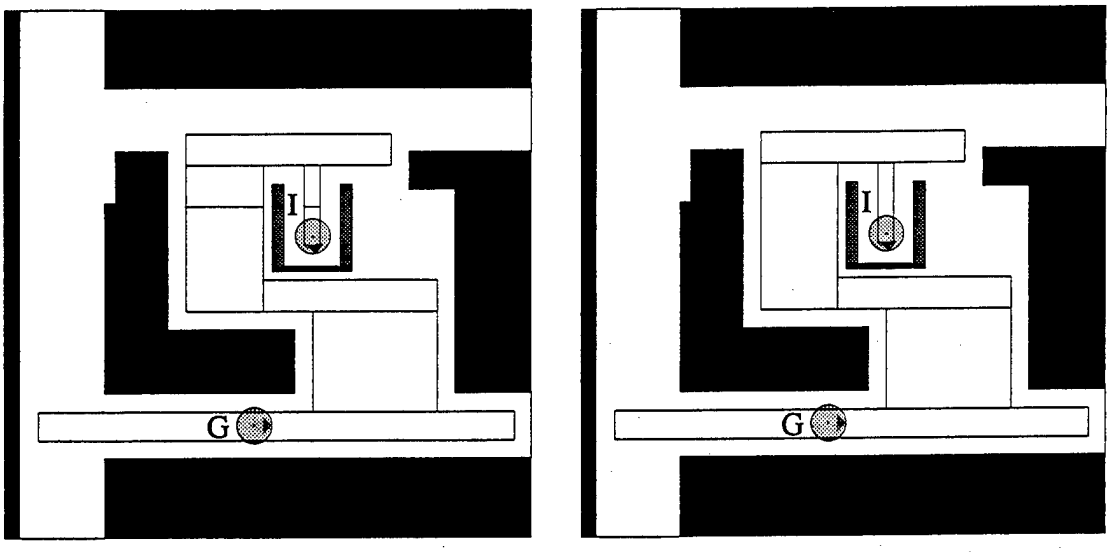


(c) decomposed cells



(d) new channel

Figure 5.3: Alternative channel generation



(a) channel (b) merged channel

Figure 5.4: Merging cells in alternative channel

two cells (the second and third cells) in the original channel have been refined, and therefore the new channel is divided by the boundary (i.e., access gate) of the current cell (a). Cells in the new channel are merged, and the resulting channel consists of less number of cells (b).

Local replanning failures can be handled by generating an alternative channel, as shown in the above examples, regardless of whether the current channel is completely obstructed or not. However, the alternative channel generation can be computationally expensive, especially when the connectivity graph is large. Therefore, we extend below the local replanning method to multiple cells. The extended replanning method attempts to find an escaping path over two or more cells. This additional technique can increase the efficiency of the navigation system.

5.3 Multi-cell Local Replanning

Multi-cell local replanning is an iterative local replanning over a sequence of cells, backtracked from the current cell. Starting from a sequence of two cells, i.e., the

current cell and the previous cell, the sequence of cells is expanded by adding one more previous cell every time local replanning fails to find a path in the current sequence of cells. Before applying multi-cell local replanning, the navigation system must determine whether the current cell (or the sequence of the cells currently considered) is completely obstructed or not. Indeed, if it is completely obstructed, the only way to proceed is by generating a new channel. In the following, we describe the process to check a cell (or a sequence of cells) for obstruction.

5.3.1 Detection of Obstruction

A channel is obstructed if any cell or sequence of cells is obstructed. A cell is obstructed if there exists no path between its access gate (or the initial configuration if the cell is the first cell in the channel) and its exit gate (or the goal configuration if the cell is the last cell in the channel). This simple definition extends to a sequence of cells.

When a robot enters a cell, it crosses the access gate of the cell at some configuration, called the **access configuration**. This configuration is recorded for each cell. Failure of local replanning in a cell implies that there is no path between the local-minimum configuration and the exit gate of the cell. This also means that there exists no path between the access configuration and the exit gate because the local-minimum configuration is attained through the access configuration.

Initially, when no unexpected obstacles have been detected, all configurations of the robot on the access gate are believed contiguous, i.e., it is expected there exists a path between every pair of configurations on the access gate. If all configurations on the access gate remain contiguous after the local replanning has failed, then it can be concluded that there exists no path between the access gate and the exit gate. Therefore, the cell (and the channel) is obstructed.

When unexpected obstacles are detected on (or near) the access gate, the configurations on the access gate are divided into separate regions (usually, two or three),

each of which consists of a set of contiguous configurations. One of the regions contains the access configuration. When local replanning fails, it is already known that there is no path between this region and the exit gate. Then, in order to determine that the cell is obstructed, it remains to check that there exists no path between any of the other regions and the exit gate. This is achieved by repeating the path planning part of the local replanning method, with the local-minimum configuration replaced by a configuration in each of these other regions. We describe below how to identify these regions.

First, the grid points corresponding to the configurations on the access gate are identified, and put into a list $\mathbf{T}$, i.e., $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n_g})$ where n_g is the number of the grid points on the access gate. The grid points in $\mathbf{T}$ are ordered such that $\boldsymbol{x}_1$ is a grid point corresponding to one end of the access gate, while $\boldsymbol{x}_{n_g}$ corresponds to the other end, and two successive grid points on the access gate are also successive in $\mathbf{T}$. The grid point corresponding to the access configuration is also identified, and it is denoted by $\boldsymbol{x}_a$.

For any i ($i \in [1, n_g]$), $\boldsymbol{x}_i$ is defined to be **free** if none of the configurations of the robot corresponding to $\boldsymbol{x}_i$ overlaps with detected obstacles. For all i ($i \in [1, n_g - 1]$), $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i+1}$ are defined to be **freely connected** if both $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i+1}$ are free. For any i and j ($i \neq j$, $i \in [1, n_g]$ and $j \in [1, n_g]$), $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are freely connected if all successive pairs between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are freely connected. Then, identifying separate regions is equivalent to splitting $\mathbf{T}$ into sub-lists so that all grid points in each sub-list are freely connected, but grid points in one sub-list is not freely connected to grid points in another sub-list.

The list $\mathbf{T}$ is split into several sub-lists. The splitting operation starts by creating a sub-list $\mathbf{T}_a$, which contains $\boldsymbol{x}_a$ initially. Then, $\mathbf{T}_a$ is built by removing all the grid points of $\mathbf{T}$, that are freely connected to $\boldsymbol{x}_a$, and by adding them to $\mathbf{T}_a$. If all remaining grid points in $\mathbf{T}$ are not free, the cell (thus, the channel) is immediately determined to be obstructed. Otherwise, the remaining free grid points in $\mathbf{T}$ are put

into other sub-lists T_i $(i = [1, n])$:

$$\begin{aligned} T_1 &= (x_{l_1}, \ldots, x_{r_1}) \\ T_2 &= (x_{l_2}, \ldots, x_{r_2}) \\ &\vdots \\ T_n &= (x_{l_n}, \ldots, x_{r_n}) \end{aligned}$$

where $l_1 = 1$, $r_n = n_g$, $l_1 < r_1 < l_2 < \ldots < l_n < r_n$, and all grid points in T_i are freely connected.

The sub-list T_i corresponds to a region that consists of contiguous configurations on the access gate. If there exists no paths for a configuration corresponding to any grid point in each of $\mathbf{T}_i$'s, the channel is obstructed.

The channel may also be obstructed by unexpected obstacles placed over multiple cells, in which case the above process is extended to a sequence of cells. A sequence of cells is considered obstructed if there is no path between the access gate of the first cell in the sequence and the exit gate of the last cell in the sequence. Therefore, the access gate of the first cell is divided into separate regions, and then each of these regions is checked for a path to the exit gate of the last cell.

5.3.2 Multi-cell Backtracking

Once the current cell is determined not obstructed, two cells (the current cell and the one previous cell) are considered for replanning. The grid is expanded so that it covers both the current cell and the previous cell. Then, local replanning is performed again over the new grid (with the same initial configuration, which is the local-minimum configuration, and the same goal configuration, which is the configuration on the exit of the cell).

If a path is found, the robot escapes the local minimum exactly as in the previous chapter. If local replanning fails again, the sequence of the two cells is checked for obstruction. If it is not obstructed, local replanning is repeated with a grid expanded

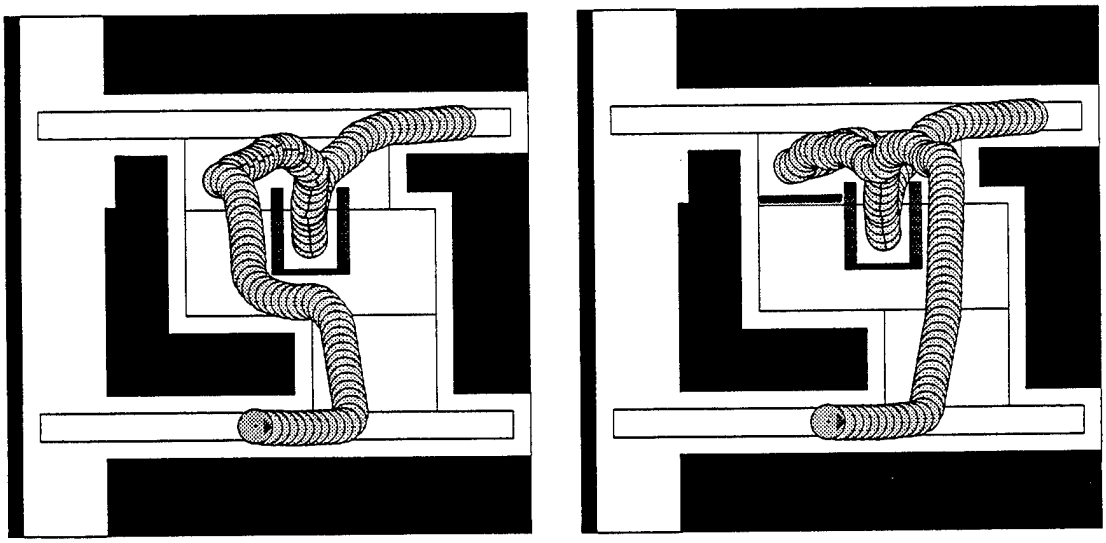
to cover one more previous cell. This multi-cell replanning technique is repeated until either a path is found, or the channel is determined obstructed, in which case an alternative channel must be generated.

However, as more cells are added for local replanning, the number of grid points becomes larger, and the local path planner takes more time. If local replanning fails over too many cells, it may become more efficient to generate an alternative channel. On the other hand, if the navigation system decides to generate an alternative channel too early, it may miss the chance of finding a simple escape route in the current channel.

In our implementation, multi-cell local replanning is attempted when local replanning fails. When the size of the grid becomes so large that the expected cost of local replanning exceeds the expected cost of generating an alternative channel, local replanning is stopped, and the generation of an alternative channel is attempted. If an alternative channel is found, the robot navigates in the new channel.

5.4 Examples

Figure 5.5 shows a robot escaping a local minimum using the multi-cell replanning. Figure (a) shows the path followed by the robot in the setup of Figure 5.2. The robot escapes the local minimum in the dead-end along the escape route, which is generated by the local path planner over two cells. Figure (b) shows a similar example, but with an additional unexpected obstacle placed over the previous escape route. The robot escapes the local minimum due to this obstacle along a new escape route. The robot does not fall into the first local minimum again because the obstacles were detected earlier and the information is used in finding the new escape route.



(a) (b)

Figure 5.5: Escaping a local minimum by the multi-cell replanning.

Chapter 6

Computer Simulation and Experiments

We implemented three versions of the navigation system described in the previous chapters, with a simulated robot, Robotworld and GOFER, respectively. The computer simulation system has been mainly used for developing and testing the algorithms, and for the study and selection of various parameters used in the navigation system. The Robotworld system was used to demonstrate the navigation system for a holonomic robot that can translate and rotate freely in the plane. GOFER was used to demonstrate the system in a real mobile robot environment using real proximity sensors to detect unexpected obstacles. The GOFER robot is also subject to nonholonomic kinematic constraints. Hence, it raises additional issues not addressed in the previous chapters. The techniques used to deal with these issues in our implementation are described below.

In this chapter we report on these three systems and we show experimental results obtained with them. These results show that our reactive navigation system deals with unexpected obstacles gracefully in physical world as well as in simulated environments.

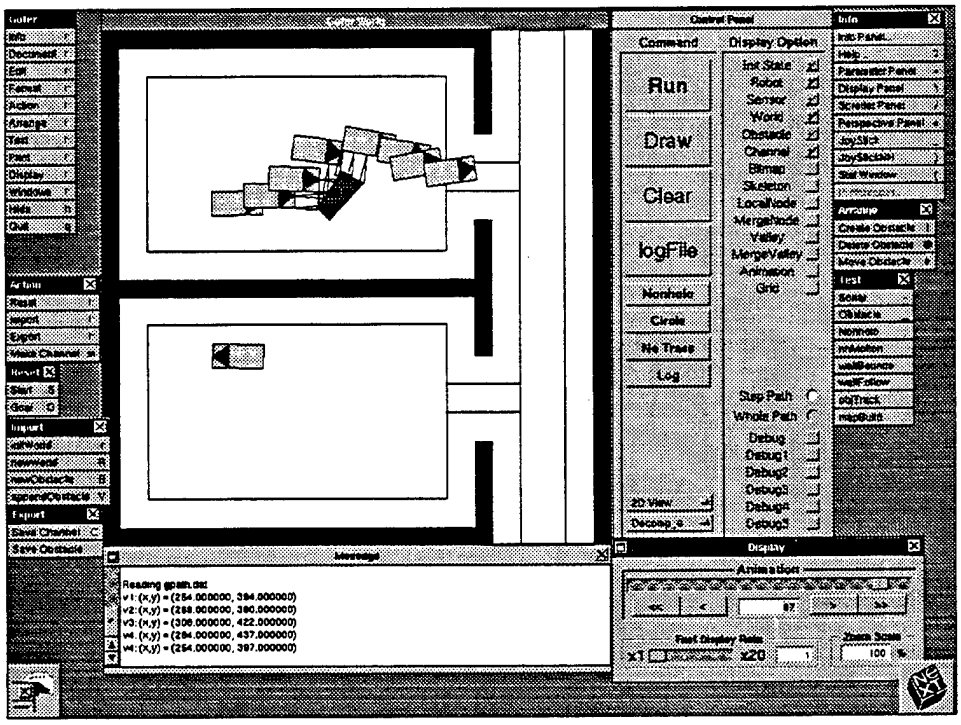


Figure 6.1: Navigation system on NeXT computer

6.1 Simulated Robot System

The navigation system based on the reactive architecture and techniques described in the previous chapters has been fully implemented. Figure 6.1 illustrates the multi-window graphic interface of this implementation on the NeXT computer. The planner (channel generation) is written in Common Lisp [ZL91]; the rest is implemented in C.[1] The system can be run to control a simulated robot. In the simulation mode, the navigation system has complete knowledge of the workspace, which includes known and unknown obstacles, but the planning component only uses the information about the known obstacles to generate the channel. The reactive component of the navigation system can only access simulated sensory data. Sensor simulation makes use of the complete knowledge of the workspace.

In simulation, we have experimented with a holonomic robot, i.e., a robot that is

[1]Graphic interfaces implemented on the NeXT computer are written in Objective-C.

not constrained by any kinematic constraint involving velocity parameters. Results with such robots have been quite satisfactory; the robot found its way to the goals along reasonably good paths (see Figures 3.6 and 4.12).

The navigation system, especially various potentials used in the reaction component, depends on several parameters. Although intuitive/qualitative reasoning about these parameters is possible, selecting a "right" set of values for these parameters, to generate a good behavior of the robot for many different cases, can be rather difficult. Numerous navigation experiments were performed, using the simulation system, with various workspace and arrangements of unexpected obstacles, until we have found values for the parameters producing "reasonable" paths for most cases. Then, additional experiments have shown that the navigation system is robust to small changes in these parameters in the sense that the robot succeeds to reach the goal even though the paths may vary.

Another difficulty in mobile robot navigation is caused by sensing errors. Data returned by proximity sensors usually combine range error and direction error (due to the rather wide beam angle of the emitted signal). To analyze the robustness of the navigation system to sensing uncertainties, we have simulated sensing errors in the simulation system by introducing random errors into the sensed range data and by using various conic beam angles. The random range errors are bounded (by some percentage of the maximum sensing range), and they are assumed to be proportional to the detected range data. Tables 6.1, 6.2, and 6.3 show simulation results when navigation was performed in the workspace of Figure 3.6 (b). We have also tested with several different arrangements of unexpected obstacles, and the results were similar.

Table 6.1 shows the simulation results when the navigation was performed with various range errors, but with a fixed beam angle (22.5°). The number of control loops represents approximately the length of the resulting path. The robot failed to reach the goal when the range errors exceeded 30% of the maximum sensing range. The erratic sensory data caused a local minimum in the third cell of the channel (see Figure 2.4 for the channel), and local replanning failed because the false obstacle grid points blocked the passage. Most real infra-red proximity sensors have less errors.

Range Error	Number of Steps	Result
No Error	864	Success
10%	870	Success
20%	879	Success
30%	894	Success
40%	953	Failure

Table 6.1: Navigation with sensing range errors

Angle	Number of Steps	Result
0.01°	845	Success
5.0°	849	Success
10.0°	851	Success
22.5°	864	Success
30.0°	869	Success
45.0°	880	Success

Table 6.2: Navigation with various conic beam angles of sensor

Proximity sensors using ultra-sound may cause problems because the range error become fairly big for obstacles at a relatively long distance.[2] Also, they are known to often produce spurious data when the angle between a sensing direction and the boundary of a detected obstacle is much different from 90°.

Table 6.2 shows the simulation results when the navigation was performed with various beam angles, but with no range errors. The changes in the angle didn't have much impact on the resulting path when unexpected obstacles are sparsely scattered. But, when unexpected obstacles are arranged close to each other, the robot stopped at many places and relied on local replanning because of the false local minima due to the sensing errors.

Finally, Table 6.3 shows the simulation results when the navigation was performed

[2]Failures due to this error can be reduced by limiting the maximum range.

Angle	Range Error	Number of Steps
0.01°	10%	852
	20%	865
	30%	870
10.0°	10%	860
	20%	871
	30%	889
45.0°	10%	889
	20%	903
	30%	924

Table 6.3: Navigation with sensing range errors and various conic beam angles

with both types of errors. Navigation terminated successfully for all cases shown in the table. The resulting paths became longer as the combined errors increased.

6.2 Robotworld Experiments

We used Robotworld [Sch87] to conduct additional experiments with the navigation system in the case where the robot translates and rotates freely in the plane (holonomic robot). The experimental setup is depicted in Figure 6.2.

Our Robotworld is made of two simple manipulation robots (I and II) and a vision system. Each robot consists of a base and a gripper. The base hangs from the ceiling and translates in the horizontal plane (two degrees of freedom). The gripper translates along a vertical axis and rotates about that axis. Therefore, each robot has four degrees of freedom in total. The vision system consists of a camera, looking downward, mounted on a translating base identical to that of a robot.

The "robot" that our navigation system controls is an object held by the gripper of Robotworld's robot I. It is made of Lego pieces. This "robot" navigates in a workspace built with other Lego pieces (some are known obstacles, the others are unexpected ones) mounted on a horizontal platform. Robot I moves the "robot"

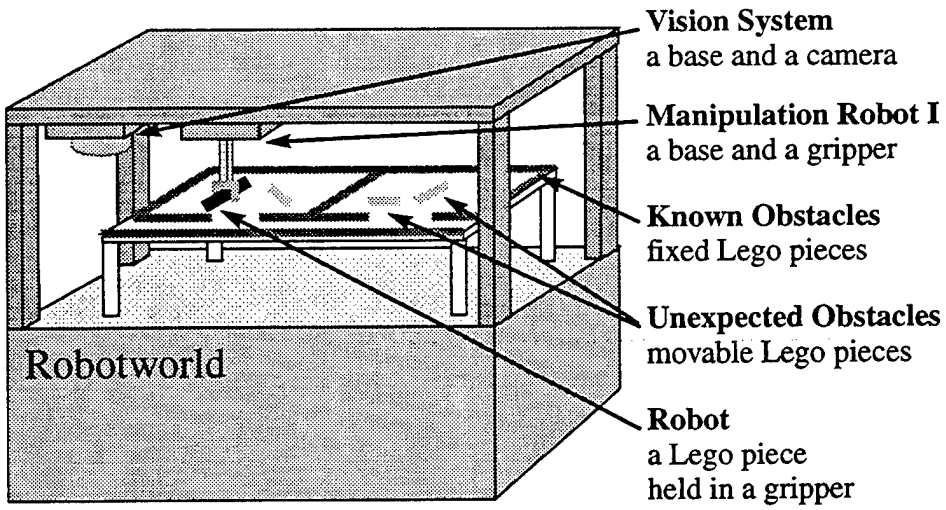


Figure 6.2: Robotworld setup

by holding it a fraction of an inch above this platform, so that it can collide with the obstacles. Since the gripper can translate and rotate freely, the "robot" is free of kinematic constraints. (Actually, wires prevent the gripper from rotating indefinitely; when a stop of the revolute joint is attained, the gripper lifts the "robot" above the obstacles, rotate back to maximize the angular distance to the revolute joint stops, and translates the "robot" down back to its previous position and orientation.)

The layout of both the expected and the unexpected obstacles is input on the computer display. (In our experiments, only the layout of the unexpected obstacles was changing.) Robotworld's robot II automatically mounts the obstacles on the workspace platform from this layout description. When this is done, robot I brings the "robot" to a specified initial configuration. After the goal configuration is input, the navigation system controls the motion of the "robot".

The vision system was used for identifying and locating obstacles in an obstacle-supply bin, and for verifying the correct placements of the obstacles in the workspace. However, Robotworld does not provide a sensing system that could be used to emulate the "robot"'s sensors and detect unexpected obstacles. Such sensing is therefore simulated using the known layout of the workspace, as in the simulation system presented in the previous section.

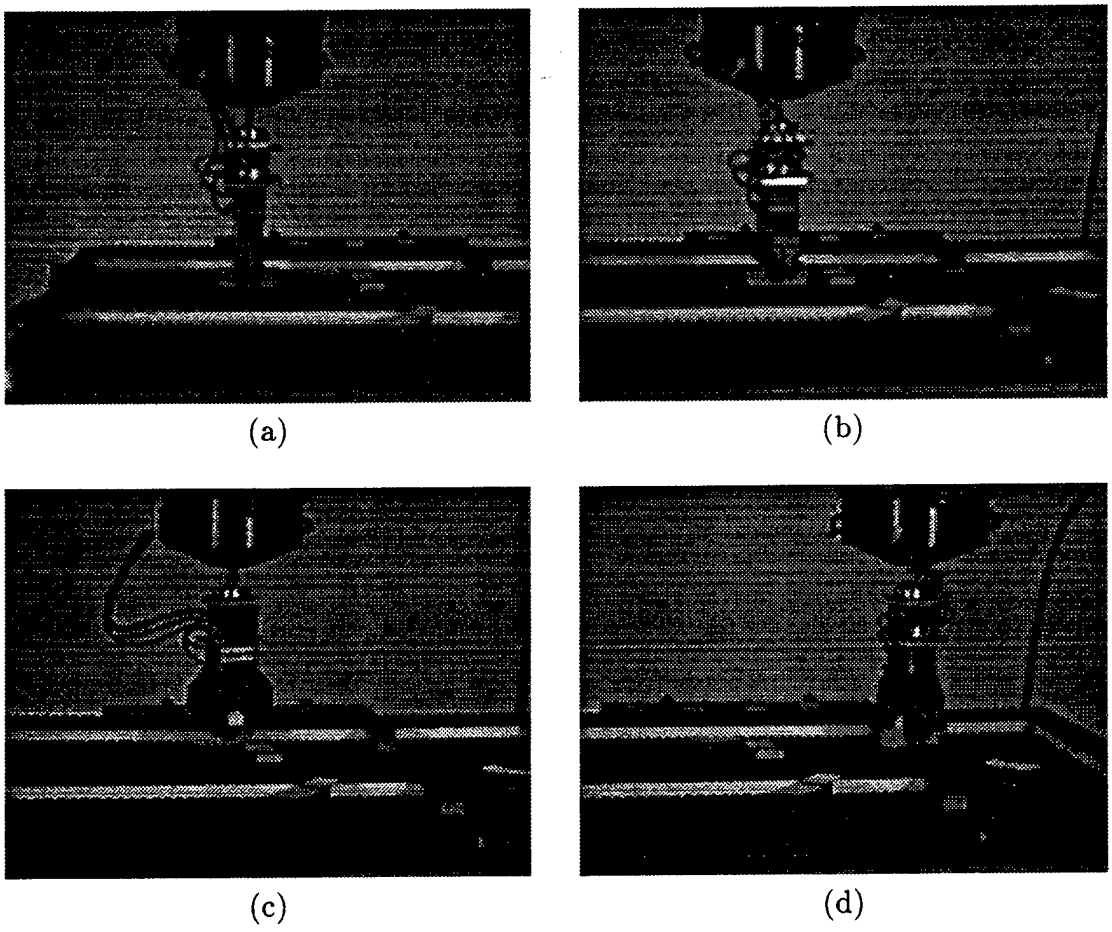


Figure 6.3: Navigation of a simulated robot experimented on RobotWorld [Over]

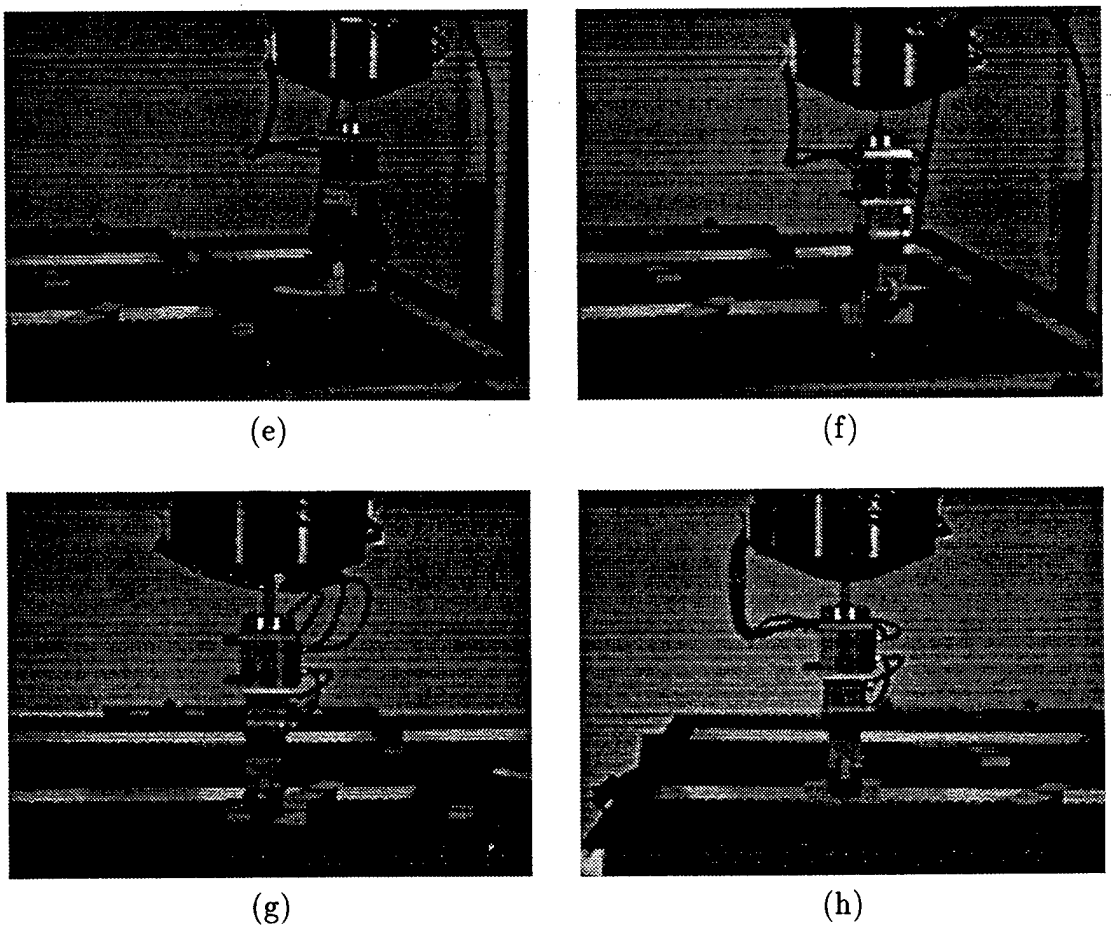


Figure 6.3: Navigation of a simulated robot experimented on RobotWorld

In this experimental setup, our navigation system runs on a DEC 3100 workstation. This system sends its commands to a program written in RAIL (the programming language of Robotworld) and running on the Macintosh II computer controlling Robotworld. This RAIL program converts the commands sent to the "robot" into appropriate motion commands for Robot I. The connection between the DEC 3100 workstation and the Macintosh computer is a 9600 baud serial line.

Figure 6.3 shows snapshots along the path of Figure 3.6, when executed with this experimental setup.

6.3 GOFER Experiments

6.3.1 Description of GOFER

Over the last three years we have developed a mobile robot, GOFER (see Figure 6.4), which is equipped with multiple sensors and on-board computing [CCL+90].[3] This robot operates in the office-like environment of our laboratory. The main characteristics of the environment, i.e., the layout of the building and the main pieces of furniture, are known a priori; but the locations and shapes of smaller objects (e.g., chairs) are not known. Most tasks to be performed by GOFER (e.g., delivering mail) reduce to navigating from one location to another without colliding with any object.

When our navigation system controls the real GOFER, the planner runs on an off-board computer (Apple Macintosh II). The description of the planned channel is sent to the GOFER's on-board computer (Dynatem DCPU30 68030 computer board) through a radio modem. The rest of the navigation system and sensory data processing runs on the on-board computer. GOFER uses a camera-laser range sensor and a ring of infra-red (IR) proximity sensors to detect obstacles and measure distances to them. In our experiments, we only used the IR sensors. Although these are less accurate and reliable than the camera-laser range sensor, the ring of IR sensors provides

[3]GOFER has a circular outline. This fact slightly simplifies channel generation, but does not modify the rest of the navigation system significantly.

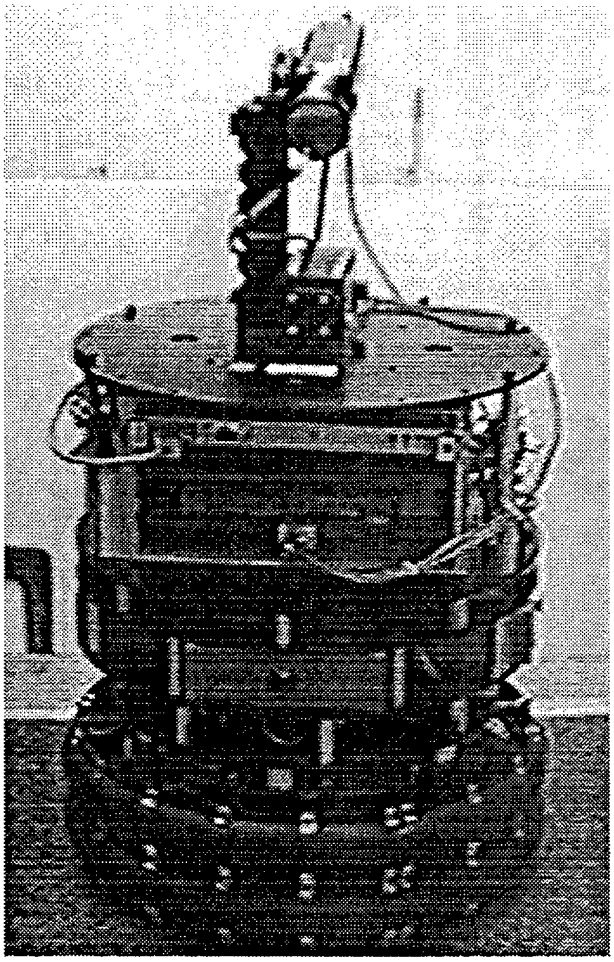


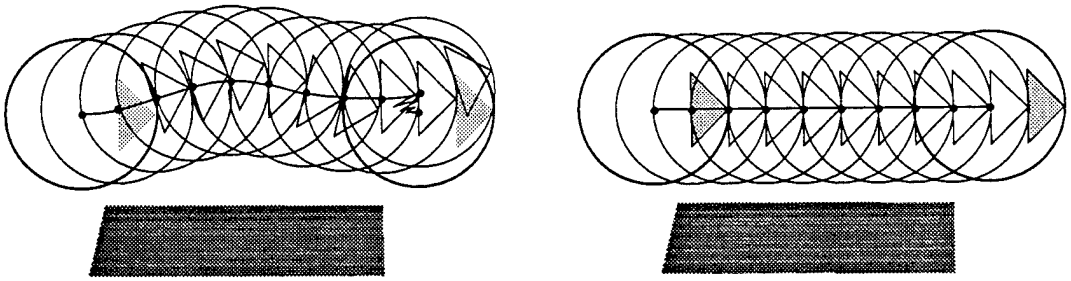
Figure 6.4: GOFER

a 360° field of view, while the camera-laser range sensor only has a 30° field of view always pointing along the current direction of motion of the robot.

6.3.2 Control of GOFER

So far, we have only considered the case of a holonomic robot (i.e., a robot that can freely translate and rotate at any time). However, GOFER is subject to a nonholonomic constraint due to its synchro-drive mechanism (see Appendix B), i.e., it can move only along the direction of its wheels at any given time. While this constraint is similar to the one constraining the motion of a car, GOFER can change its orientation without translating, i.e., it has zero turning-radius. Thus, GOFER can move from the current configuration to another by executing the following sequence of motions: stop at the current position; rotate to point in the direction of the desired position; move to the position; stop and rotate to the desired orientation. The potential-field guidance in our navigation system may be directly applicable to the control of GOFER, but the negated gradients of the potentials, in general, require GOFER to frequently change its direction of motion, especially in the presence of unexpected obstacles. This causes frequent stop-and-rotate motion sequences.

Several results have recently been reported for planning collision-free motions for nonholonomic robots in a static environment [Lau87, LTJ90, BL89] as well as in a dynamic environment where unknown obstacles or moving obstacles are present [Fra90]. The approach in [LTJ90] is based on the result that when a nonholonomic vehicle is constrained by lower-bounded turning radius (i.e., by a limited range of its steering angle), a path of minimal length between two configurations consists of a finite sequence of straight-line segments and circular arcs generated with the minimum turning radius [RS90]. However, in practice, a real robot (or a vehicle) cannot exactly follow such path because there are discontinuities of the curvature along the path and the robot cannot instantaneously change its steering angle (i.e., the corresponding turning radius) from one extreme value to another. Besides, in the presence of unexpected obstacles, the optimality in the path length becomes much less meaningful because the path needs to be modified when it overlaps with the obstacles



(a) With obstacle potential (b) Without obstacle potential

Figure 6.5: Reduced oscillatory motion

and, in the end, the actual trajectory can often be far from optimal. The approach in [Fra90] is based on using potential fields generated over a bitmap representation of the environment, but it is not applicable to our case because the computation of such potential fields takes too long to be done in real-time.

Our navigation system discretizes the range of values of the steering angle of GOFER. For each discrete value, it integrates the equations of motion over a short interval of time (with both positive and negative linear velocities), yielding possible configurations, called **candidate configurations**, where the robot can go. Among these configurations, the one with the smallest value of the potential function computed by our navigation system is selected. The corresponding steering angle and linear velocity are sent to the robot's controller for execution.

This approach produces smooth paths when the robot travels in a channel with no unexpected obstacles, i.e., when the driving command is computed from the negated gradient of the channel potential alone. However, when the robot gets close to unexpected obstacles, the unexpected-obstacle potential may cause unnecessary oscillatory motions. The cause for these motions is that motion commands are obtained from the noncontinuous candidate configurations. Since the direction of motion at all candidate configurations is known with good approximation, potential collision of the robot with detected unexpected obstacles can be predicted using this knowledge. Therefore, we further smooth the path by zeroing the unexpected-obstacle potential at the candidate configurations where the robot would not head toward detected obstacles.

The Generalized Potential Field approach, previously described in [Kro84], is based on a similar concept. Repulsive potential (due to an obstacle) is computed based on the current velocity of the robot as well as the distance to the obstacle, i.e., the value of the repulsive potential increases as the robot approaches faster and closer to the obstacle while it becomes 0 when the robot moves away from the obstacle. However, this method is less effective in reducing the oscillatory motion because only the current motion direction (rather than several possible motion directions) is taken into account.

Figures 6.5 illustrates this with an example: When all the candidate configurations are under the influence of the unexpected-obstacle potential, the robot may attempt to steer away from the detected unexpected obstacle even when the robot may travel parallel to the obstacle without hitting it (a); When the unexpected-obstacle potential is ignored for the candidate configurations that do not result in the collision of the robot with the obstacle, the robot does not oscillate but moves straight to its goal (b).

6.3.3 Experimental Results

The above control technique has given very good experimental results in simulation (see Appendix C). The experimentation with the real GOFER also gave satisfactory results, but we had some problems with the quality of the sensory data. This quality could be improved (for instance by averaging several successive readings of the sensors), but we were limited by the fact that all the software runs on the same 68030 processor. This processor alone is too slow to support both the computation of the potentials and the direction of motion at a reasonable rate (10-20 Hz) and an adequate processing of the incoming sensory data. This difficulty could be resolved by adding an on-board processor to treat sensor inputs. Figure 6.6 show six snapshots along a path executed by GOFER among unexpected obstacles. Figure 6.7 show GOFER escaping from a local minimum created by three unexpected obstacles.

The main limitation of the current system, when connected to GOFER, comes from the fact that it relies exclusively on the odometric sensors to determine the robot's

current configuration (which is needed to compute $\mathbf{U}_c$). Hence, the system becomes unreliable when the wheels slip or slide on the floor. But, the robot's current configuration can be estimated more accurately by various localization techniques, e.g., matching environmental sensory data against the workspace model, relying on self-contained inertial navigation system, or relying on global positioning system (GPS) (see Section 1.7.2 for references).

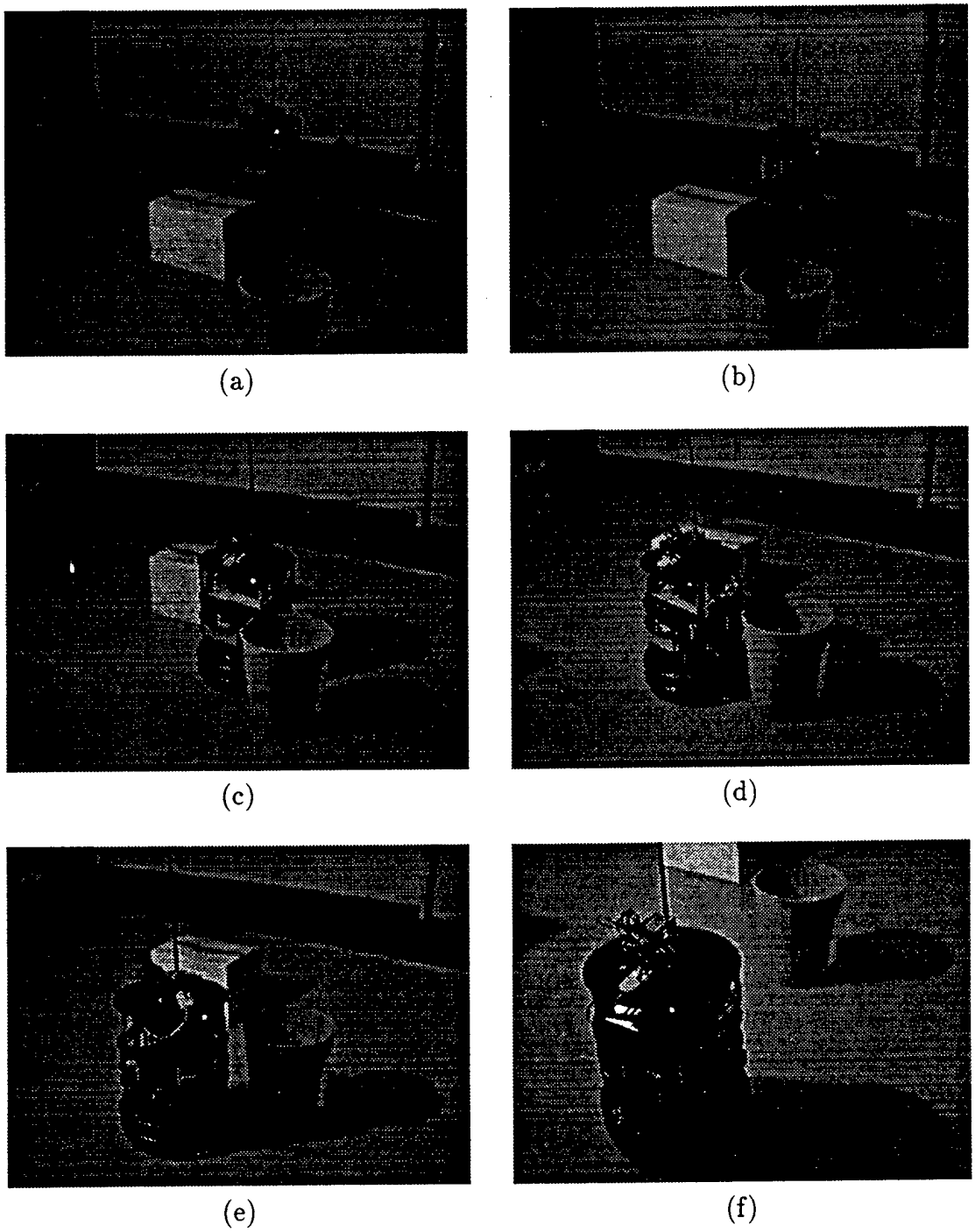


Figure 6.6: Navigation of GOFER among Unexpected Obstacles

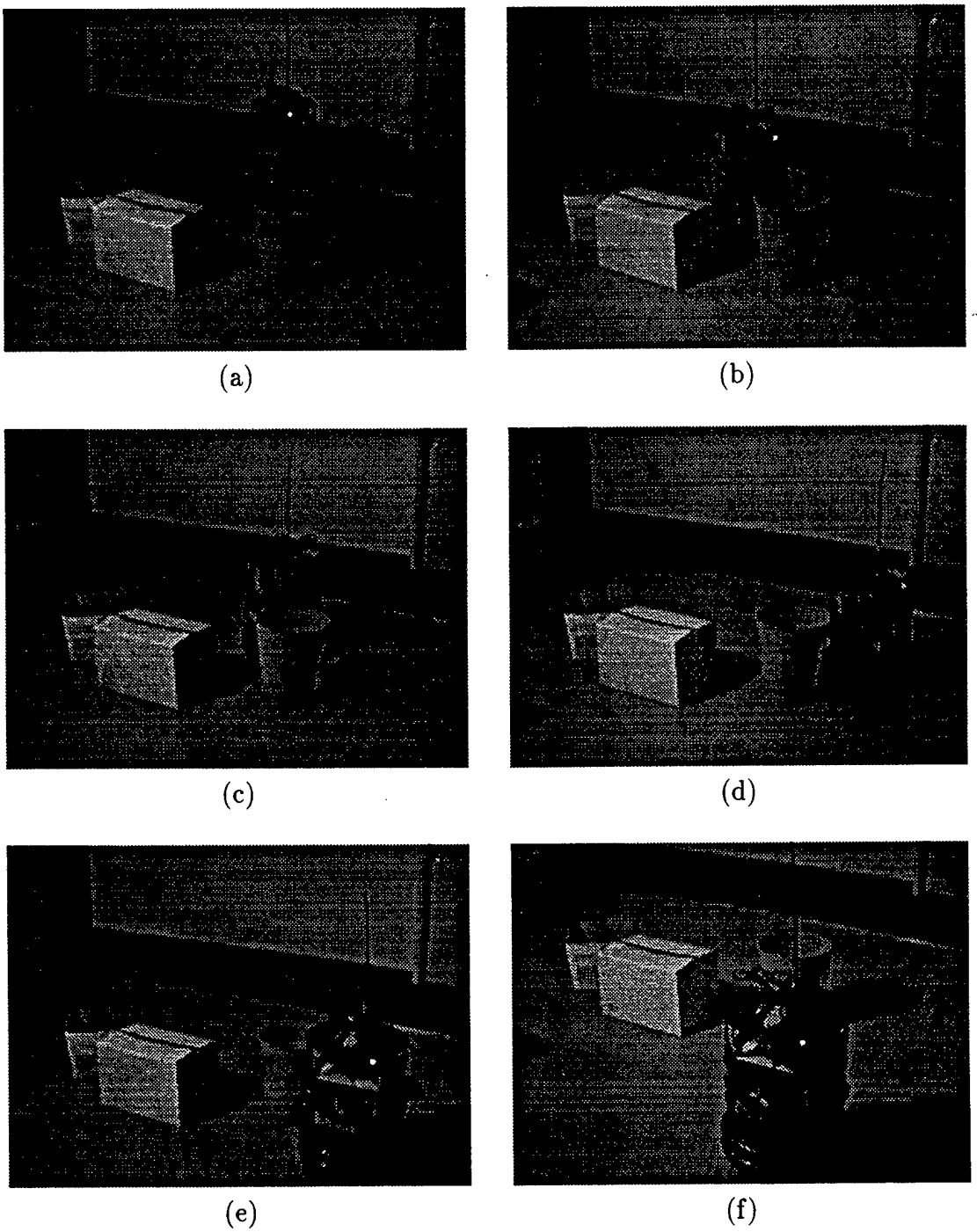


Figure 6.7: GOFER escaping a local minimum

Chapter 7

Conclusion

7.1 Summary of Contribution

We addressed the problem of developing a navigation system for mobile robots operating in partially known environments. We proposed a novel approach emphasizing interaction between planning and reaction components. We developed computational techniques to implement this approach into an operational navigation system. Three version of this system were implemented and experimented with simulated and real robots. The navigation system is complete under the assumptions that all obstacles are stationary and sensing is perfect. Under these two conditions, if it is possible to attain the goal, the robot will ultimately reach it, otherwise it will eventually give up. The experimental results show that this new approach achieves increased robustness in the presence of unexpected obstacles.

This research brings two levels of contributions: (1) the reactive architecture of the navigation system; (2) the techniques embedded in this architecture.

At the architecture level, we have borrowed the concept of a reactive architecture and applied this concept to mobile navigation by combining planning and reaction components:

- The reaction component must have some global knowledge of the robot's workspace in order to react appropriately to unexpected obstacle events. This

knowledge is provided in the form of a lesser-committed motion plan generated by a planning component aware that there may exist unexpected obstacles. In our system, lesser-committed plans take two forms: channels and valley-shaped potentials.

- Multiple layers of treatment deal with classes of events according to their expected frequency. The top layer can treat alone all the events that the lower levels are intended for. The lower levels only provide more efficient treatment. This architecture makes it possible to introduce a reliable treatment of unexpected obstacles by building the top layer. This function can then be made more efficient by adding new layers. Our system consists of three layers: channel navigation, local replanning and global replanning.

At the technical level, this work brings the following contributions:

- We have introduced the concept of a channel as a lesser-commitment motion plan, and instantiated this concept as a sequence of rectangloid cells that can easily be generated using an approximate cell decomposition planning method.
- We have defined potential field functions computed on-line to navigate in a channel toward the goal and to simultaneously react to unexpected obstacles. The potential is guaranteed to be local-minima free when there are no unexpected obstacles.
- We have developed a new way to escape local minima on-line by replanning a local path and integrating it in the current potential field function using the notion of a valley-shaped potential.
- We have extended our navigation system to robots with nonholonomic motion constraints.

7.2 Directions for Future Work

The advantage of the strong interaction between the planning component and the reaction component in our navigation system has been demonstrated by our multiple

experiments with both simulated and real robots. Nevertheless, our approach and the implemented navigation system have some limitations.

Our navigation system, based on potential functions defined in channel and local-minimum escape strategy, makes the robot adaptive to the unknown obstacles (both stationary and moving obstacles). However, when moving obstacles exist in the robot's workspace, the current navigation system is no longer guaranteed to succeed, because the local/global replanning relies on the traces of the sensory data obtained from detected unexpected obstacles. In order to increase reliability, the navigation system must be able to distinguish moving obstacles from static ones (e.g., by measuring the relative speeds of detected obstacles) and only record the traces of static obstacles. Identifying moving obstacles by using proximity sensors is usually not very reliable. Therefore, in addition to proximity sensors, more capable sensors (e.g., vision sensor with image processing) are required.

As more reliable mobile robots become available, several robots may operate simultaneously in the same workspace. Our navigation system can be further extended to deal with multiple robot navigation. The division of the planning and reaction components in the navigation system makes the combination of centralized and distributed computation more feasible. The planning component can be implemented in a central dispatcher where multiple channels are planned: one channel is planned for each robot based on the task/schedule of the robot. Then, the reaction component is implemented for each robot so that obstacle avoidance is performed independently by each robot. One of the requirements of the navigation system in application to multiple robot navigation is that sensors of one robot must not interfere with those of other robots. Laser-camera range sensor described in the Appendix B is a good candidate for such non-conflicting sensors. Some "traffic rule" may also be introduced and embedded in the channel to resolve conflicts at the intersections of channels. Existing approaches to motion planning of multiple moving objects are described in [Lat91].

Appendix A

Channel Navigation Examples

In Chapter 3, we have shown an example of robot navigation in a channel where all cells range over $[0, 2\pi]$. In the following, we show more navigation examples where the robot's orientation in some cells is limited to a subset of $[0, 2\pi]$. Figure A.1 illustrates the navigation in the channel of Figure 2.3: (a) shows the path of the robot in the 2-dimensional workspace when there are no unexpected obstacles; (c) shows the path of the robot in the presence of two unexpected obstacles; (b) and (d) show the paths corresponding to (a) and (c) in the 3-dimensional channel.

Figures A.2, (a) through (n) display snapshots at various stages of the navigation process shown in Figure A.1 (b) and (d). The robot in the 2-dimensional workspace and its corresponding configuration in the 3-dimensional channel are shown in each of the snapshots. Note that, between Figures (h) and (i), the orientation of the robot changes from 2π to 0.

Figure A.3 illustrates another navigation example: (a) shows the path of the robot when it navigates in the 2-dimensional workspace of Figure 2.4 (d); (b), (c) and (d) display the corresponding path in the 3-dimensional channel from three different view points.

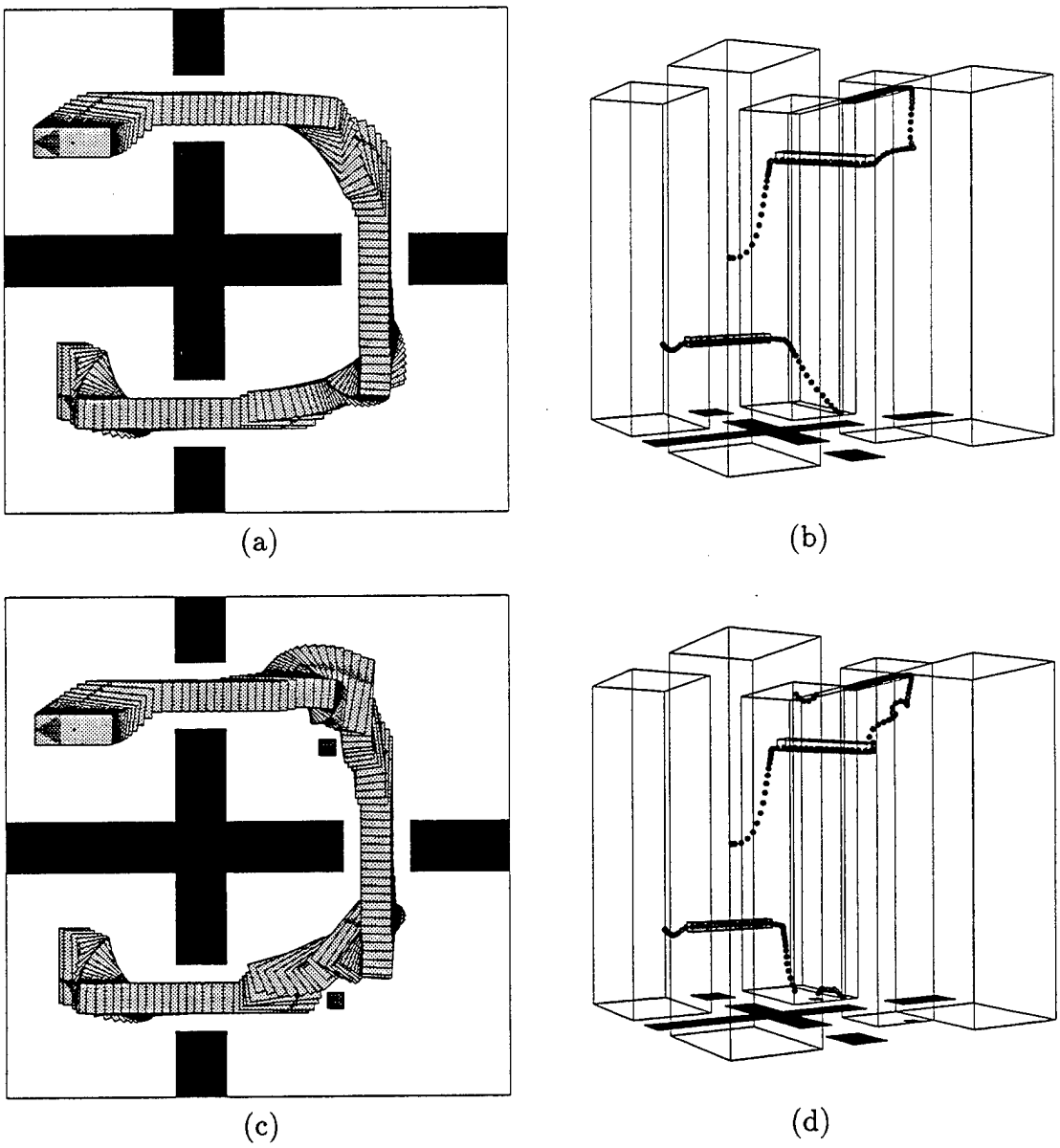


Figure A.1: Robot paths with and without unexpected obstacles

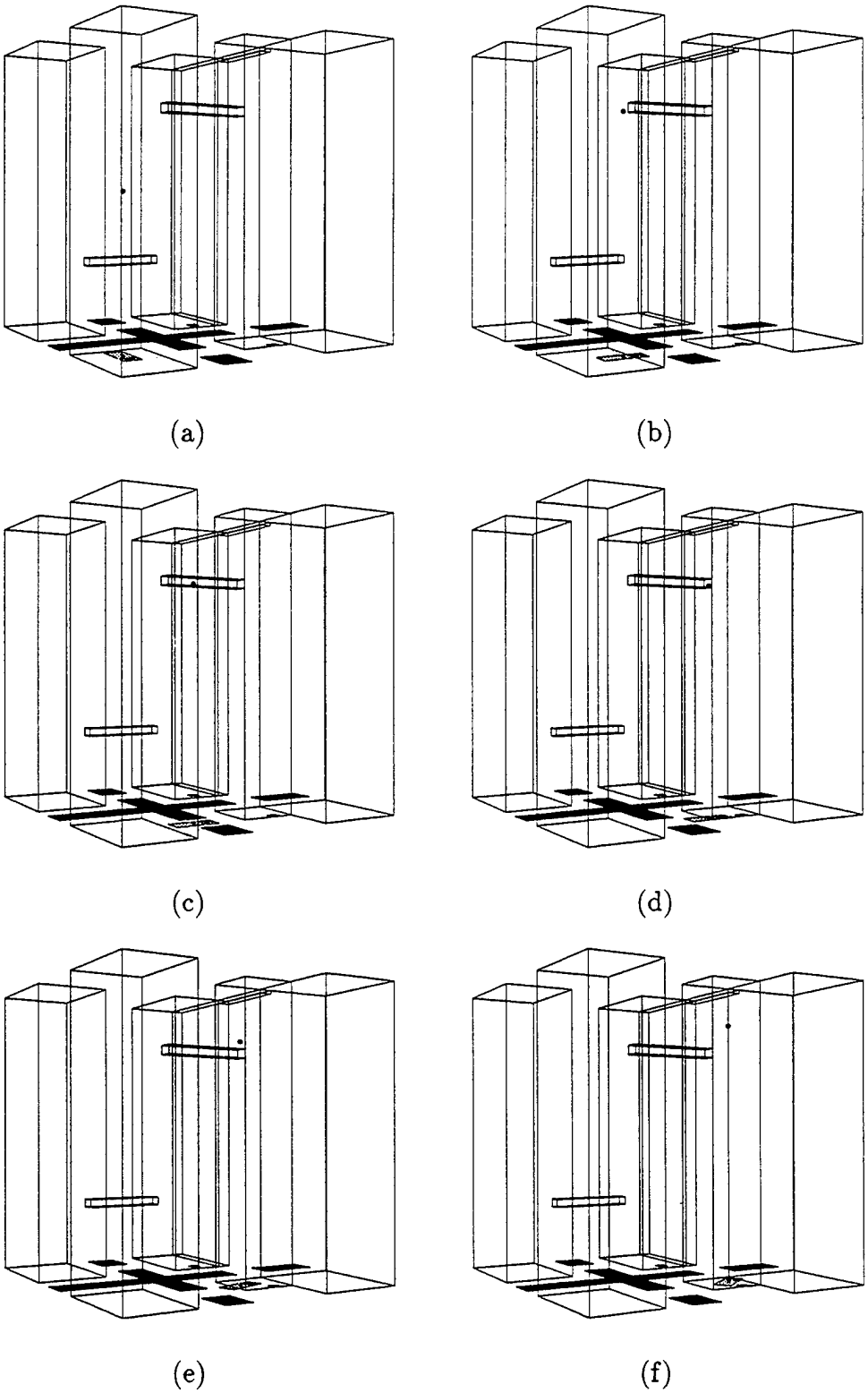


Figure A.2: Navigation of the robot shown in 2D and 3D [Over]

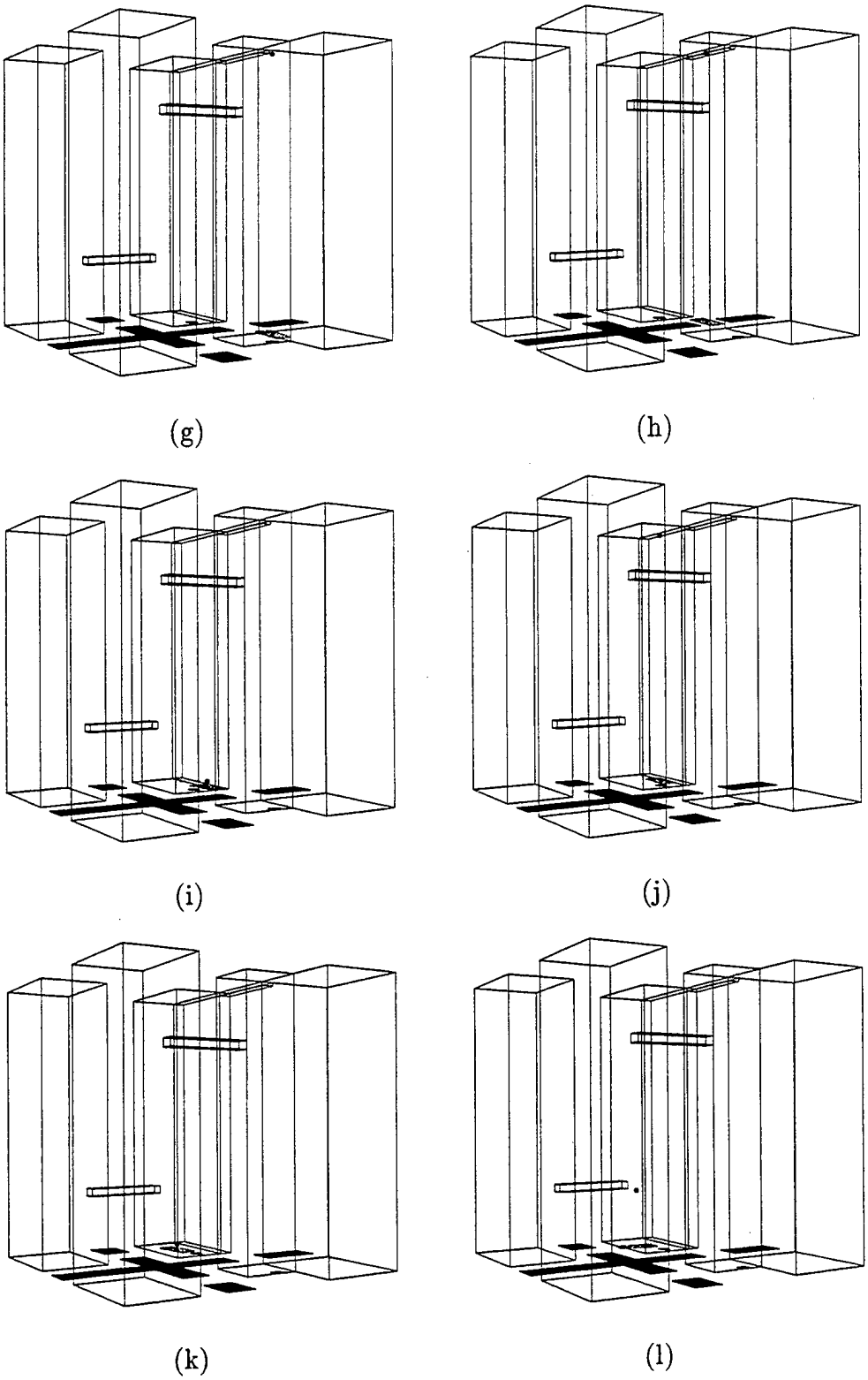
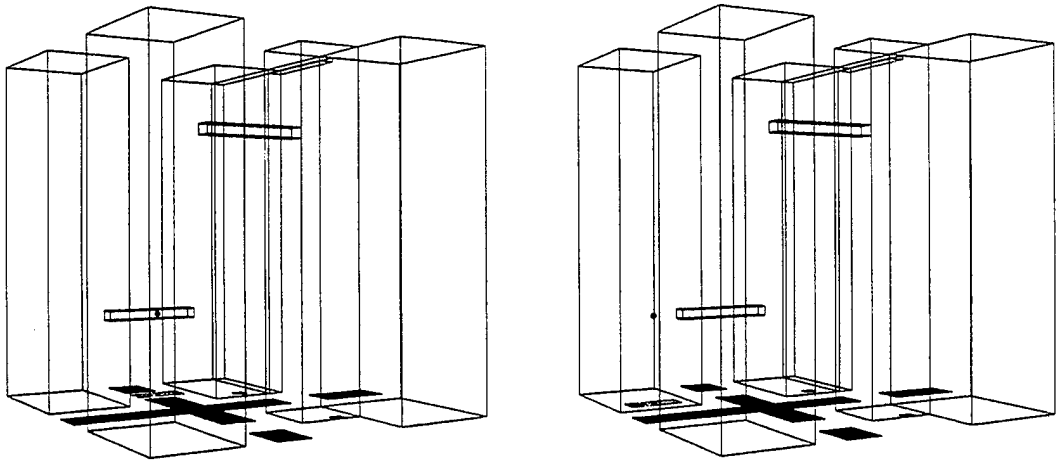
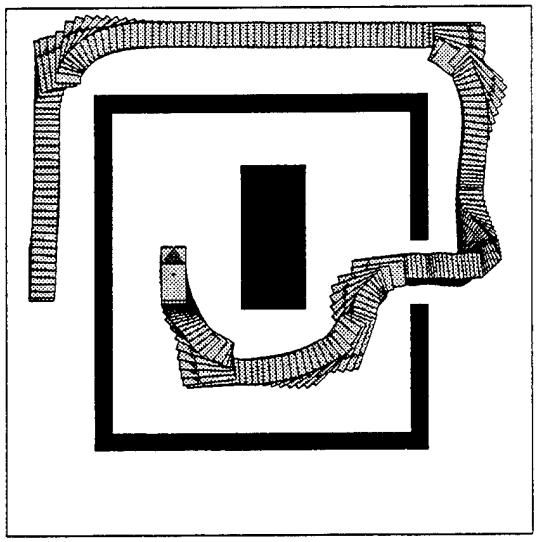


Figure A.2: Navigation of the robot shown in 2D and 3D [Over]

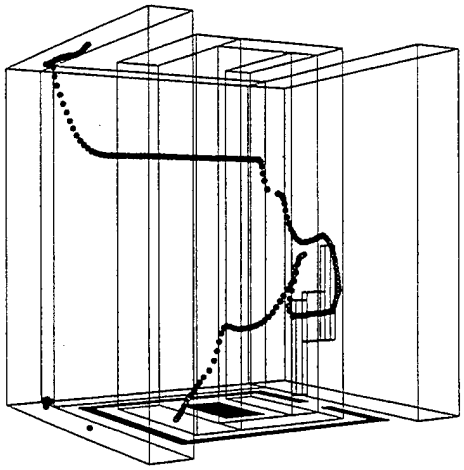


(m) (n)

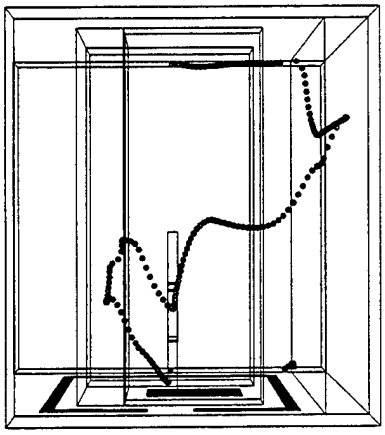
Figure A.2: Navigation of the robot shown in 2D and 3D



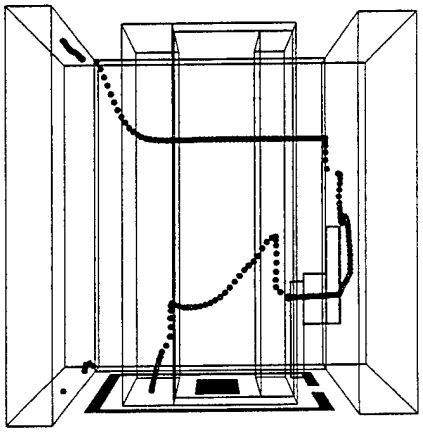
(a)



(b)



(c)



(d)

Figure A.3: Robot paths in various 3-dimensional perspectives

Appendix B

GOFER Hardware

The hardware of GOFER consists of a 12-inch diameter mobile base and interface modules (see Figure B.1). The 3-wheeled 2 DOF mobile base (B12 by Real World Interface [Rea88]) is equipped with two DC motors, four 6V gel-cell batteries and an 8-bit microcomputer for low-level control. The mobile base has a belt-driven synchro-drive mechanism which allows the base to translate and rotate independently. When the base rotates, only the top plate of the base and the wheels rotate. Two optical shaft encoders are directly attached to the motors to provide linear and angular positions of the base. Motor control and inquiries about encoders, battery status and motor status are processed by the base microprocessor. The interface modules consist of a touch sensor module, an infra-red module, a laser-camera ranging module and a computer module.

Each module, except the touch sensor, is built on its own modular plate. All modules are placed one on top of the other, with the bottom one rigidly attached to the top plate of the base. Interface modules are designed such that they are independent of each other. Also, each module communicates on a common robot bus which is passed through each module. Therefore, each module can be detached for test or repair without affecting the use of other modules. This design allows easy expandability. Independency among modules facilitated debugging and experimentation of our navigation system.

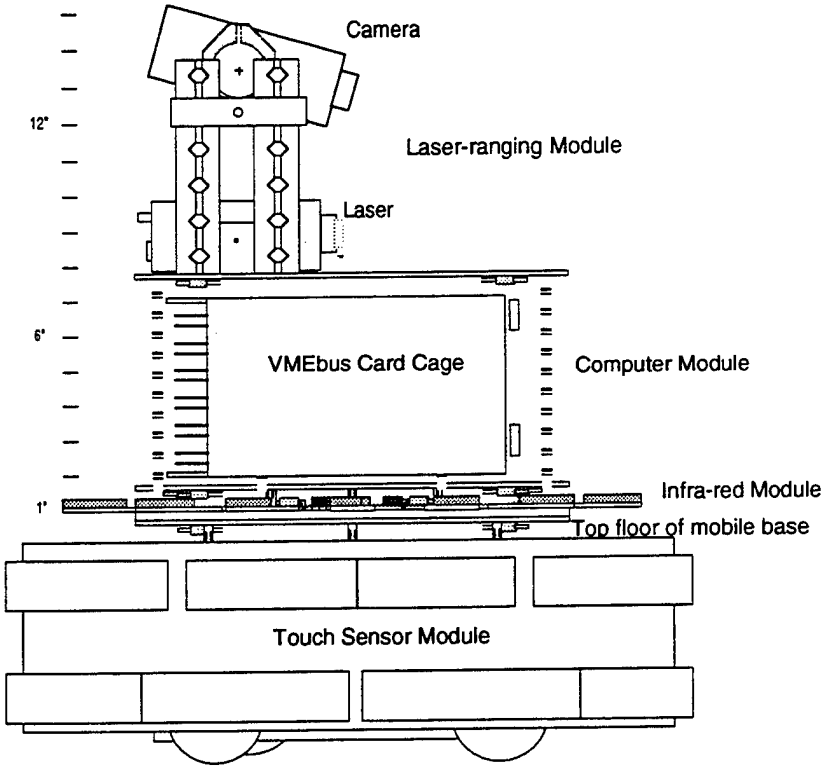


Figure B.1: Closeup of hardware modules in GOFER

The computer module has a 5 slot VME-bus card cage. A totally CMOS 20MHz MC68030 based computer board and a custom-designed IO board are installed in it. The 68030 board has battery backed 1 Mega byte SRAM and four RS232 serial ports, three of which are currently being used (one for the communications between the robot base computer, the other two for the communications between the host computer for down-loading of the system software and the input and output of the user commands). The computer board draws about 1Amp and the IO board has 64 general IO lines and draws less than 100mA.

The touch sensor module is placed around a steel bumper which encases the robot base. The touch sensor has two levels of 12 segment tape switches. Each tape switch detects a pressure of 9 oz. and provides on/off information. Therefore, the touch sensor module serves to protect the base as well as to detect the direction of contact between the robot and obstacles.

On the infra-red sensor module, 16 emitter/receiver pairs are evenly distributed around the perimeter of the modular plate and a control circuit board is located at the center of the plate. Each infra-red emitter/receiver can detect an object up to 18 inches in 250 micro-seconds with an average resolution of 0.25 inches (the accuracy increases as the range decreases). This is an intensity based device, so it is inherently subject to color and specularity problems. For our navigation system, we solved some of these problems by conservative measurement of the signal.

On the laser-camera ranging module, a laser diode and a CCD camera are placed in a reconfigurable fixture. The fixture allows the laser unit and the CCD camera to have different offsets and relative orientations. Ranging is achieved by emitting a plane of light and watching from some offset distance for intersections with this plane. Then, triangulation is done to find the actual distance. An infra-red laser diode and a cylindrical lens create the plane of light. By turning a camera 90 degrees on its side, so the scan lines run vertically, we can measure the distance to an object by monitoring the composite video output. By measuring the time from the beginning of any scan line to the sudden increase in intensity that would correspond to an object reflecting the laser light, we get a direct relation to the object's distance. An interference filter

that allows only the laser light frequency to pass is put over the camera lens to reduce the noise. This system provides 15360 data points per second, 512 ranging values over the approximately 30 degree field of view of the camera 30 times per second. The timing information along with the intensity of the reflection is then directly dumped into memory by DMA over the VMEbus. The resolution varies nonlinearly with the distance: closer objects are seen with better resolution. The resolution is directly related to the offset of the camera with the laser plane. So a 6 inch offset produces a theoretical resolution of 0.03 inches at one foot and a resolution of 2.3 inches at ten feet.

Appendix C

GOFER Simulation Results

In this appendix we show simulation results of GOFER navigation to compare the motion commands and the linear/angular velocities in three different cases. Figures C.2, C.4, and C.6 illustrates the paths navigated by GOFER in the computer simulation system for three different cases, i.e., with no unexpected obstacles, with unexpected obstacles causing no local minima, and with unexpected obstacles causing local minima. Figure C.2 (a) shows the initial and goal configurations and the corresponding channel. The same initial and goal configurations are used for the other two cases, and, thus, the channel is identical for all three cases. Figures C.3, C.5, and C.7 illustrates the motion commands (shown in (a)'s) generated by the navigation system, the resulting linear velocities (shown in (b)'s) and rotational velocities (shown in (c)'s) of GOFER during navigation.

Motion commands corresponding to the discrete turning radii are indexed such that the number 0 (resp., -1) corresponds to the forward (resp., backward) translation without rotation, and the positive numbers correspond to the clockwise rotation of the robot while the negative numbers, except -1, correspond to the counterclockwise rotation. Figure C.1 illustrates such indexing.

Figures C.3 (b) and C.5 (b) show that the robot traveled at maximum linear velocity except during the initial acceleration period.[1] However, in Figure C.7 (b),

[1]In the simulation, the execution of the navigation system has been terminated, without waiting

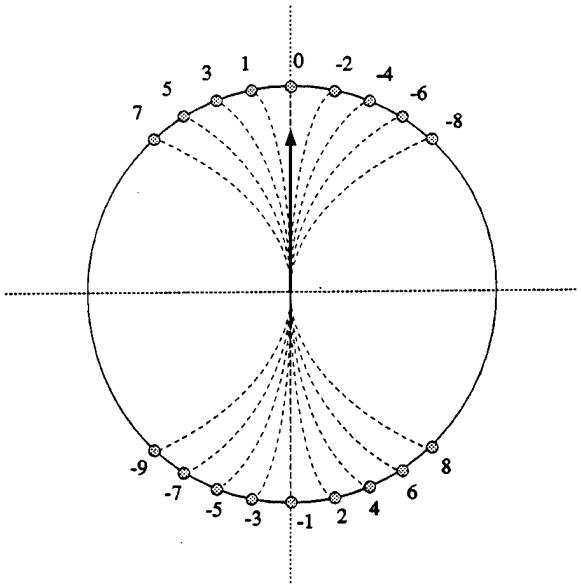
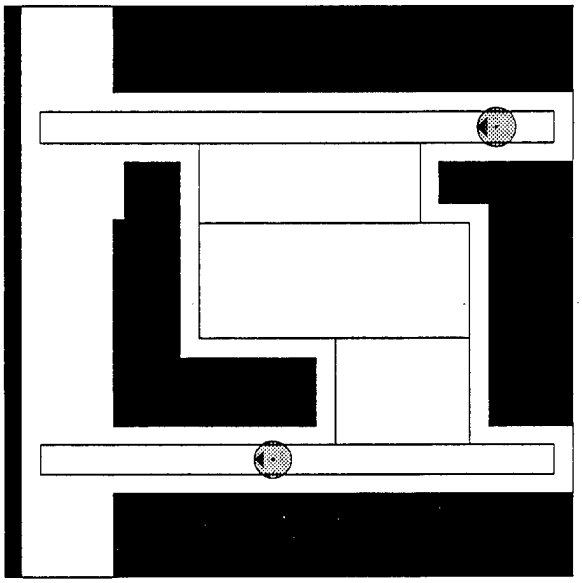


Figure C.1: Discrete turning radii and the corresponding indexes

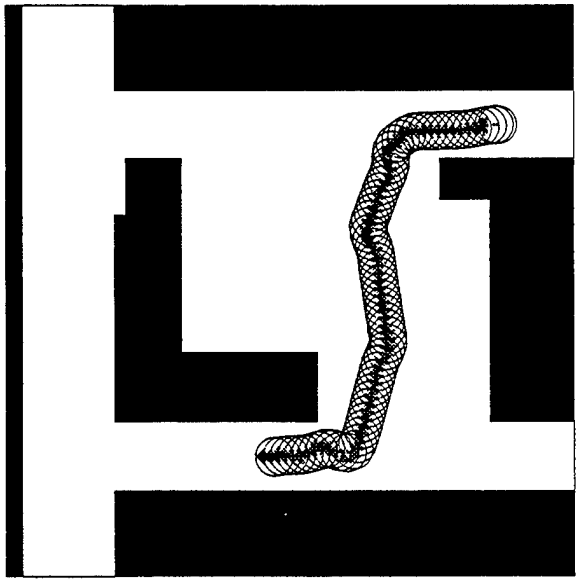
the robot slows down and stop when the local minimum is attained. It then moves backward and escapes the local minimum. When the robot is outside the attraction basin of the local minimum, it travels forward at the maximum linear velocity again. The first two glitches in the figure happened due to the local minima. The third glitch happened when the robot tried to go between two obstacles near the bottom. At first, the local-minimum "alert" flag was set due to the repulsive potentials from the two obstacles, but the robot found a way around the last obstacle (in the bottom) before the "detect" flag was set. Therefore, the robot regained its full speed without stopping, and moved around the obstacle.

The dimension of the workspace used in the simulation is 300×300 in^2. The maximum linear velocity was set to 0.013 in/sec ($\approx$ 0.1 m/sec), and the four discrete turning radii were set to be the radius of GOFER multiplied by 1.0, 2.0, 2.5, and 3.0, respectively. Each time step is 0.1 second. The trace of the robot is shown every 5 time steps.

for the robot to stop completely, when the robot reaches near the goal within a prespecified distance. Therefore, the linear/angular velocities in the Figures C.3-C.7 do not drop to 0 at the end.

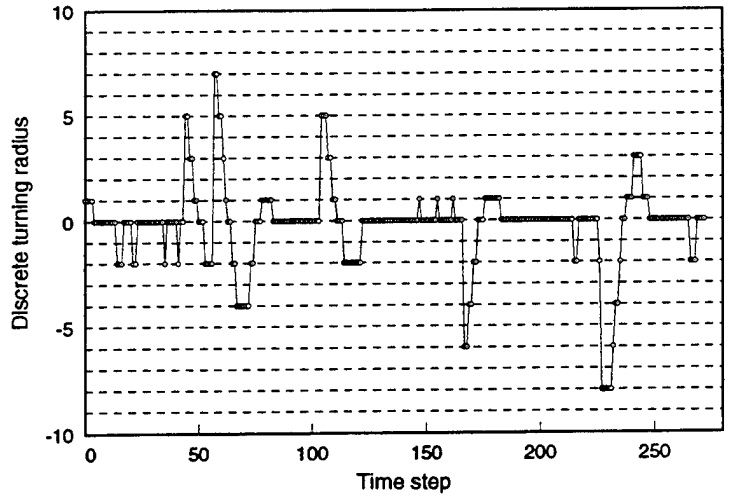


(a) Initial and goal configurations, and the Channel

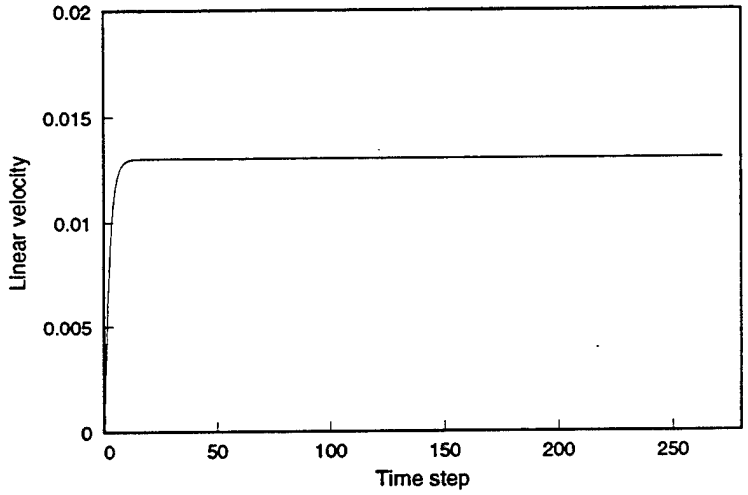


(b) Trajectory

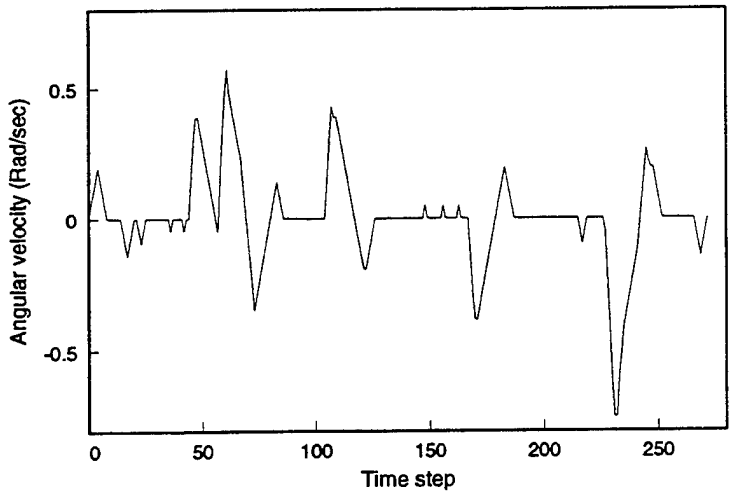
Figure C.2: Example 1: Navigation of GOFER with no unexpected obstacles



(a) Control history

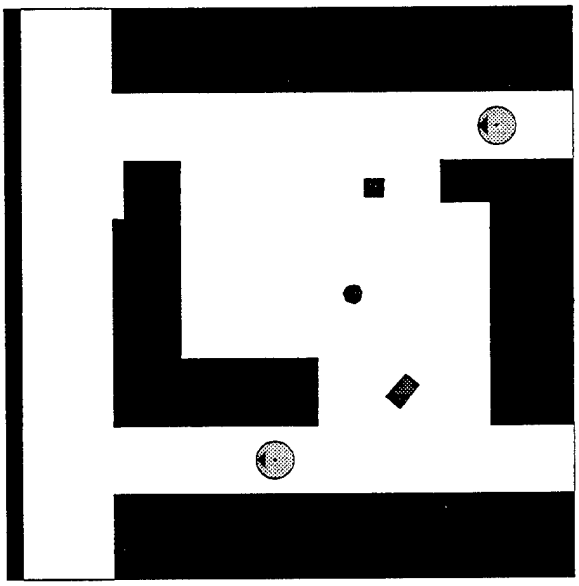


(b) Linear velocity

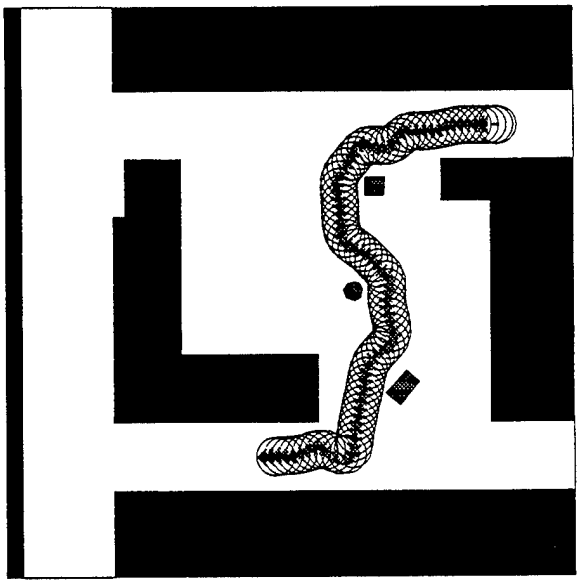


(c) Angular velocity

Figure C.3: Example 1: Control history, the linear and angular velocity profile

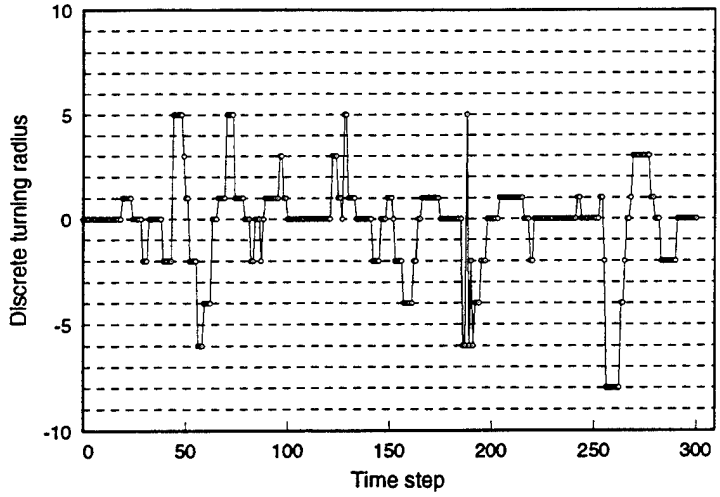


(a) Initial and goal configurations

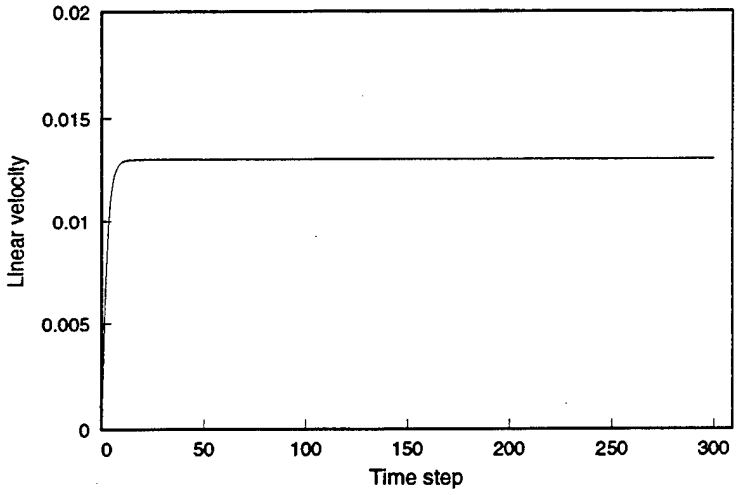


(b) Trajectory

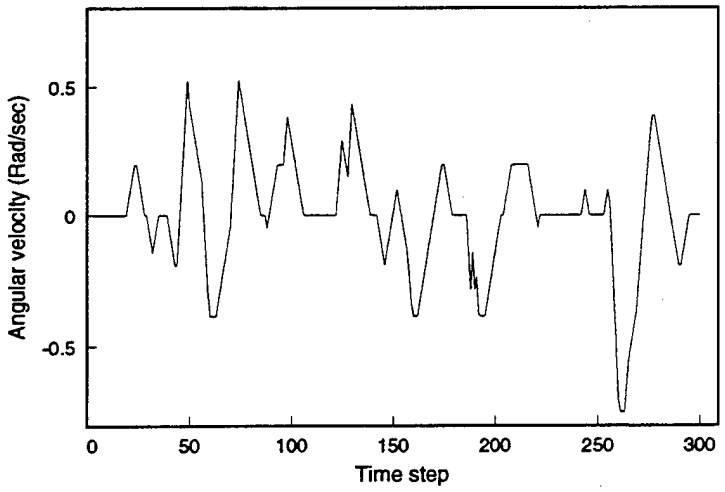
Figure C.4: Example 2: Navigation of GOFER with unexpected obstacles



(a) Control history

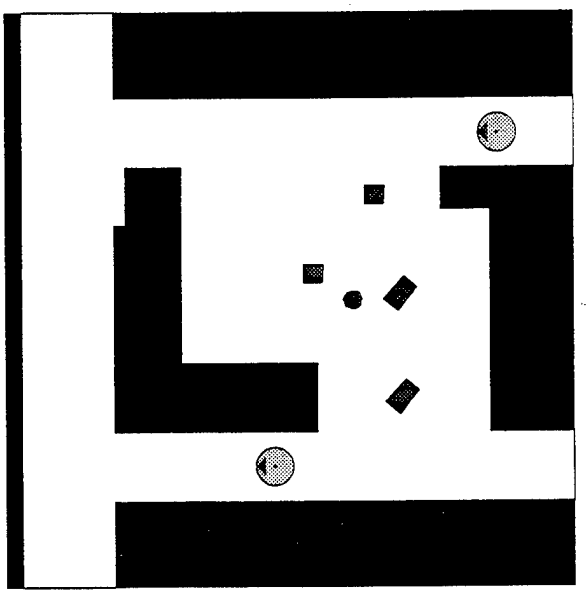


(b) Linear velocity

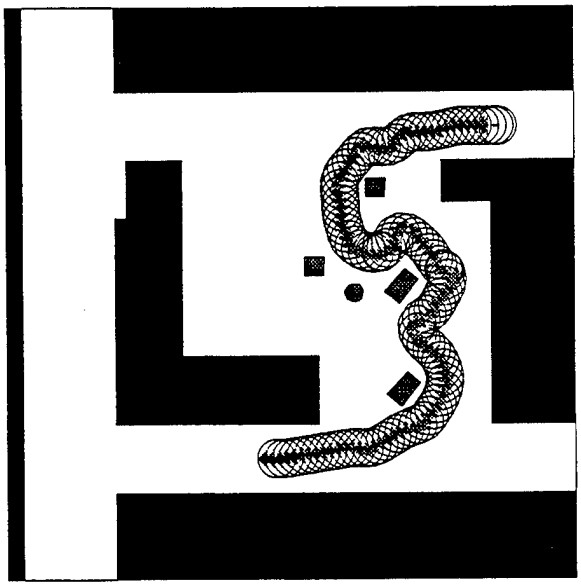


(c) Angular velocity

Figure C.5: Example 2: Control history, the linear and angular velocity profile

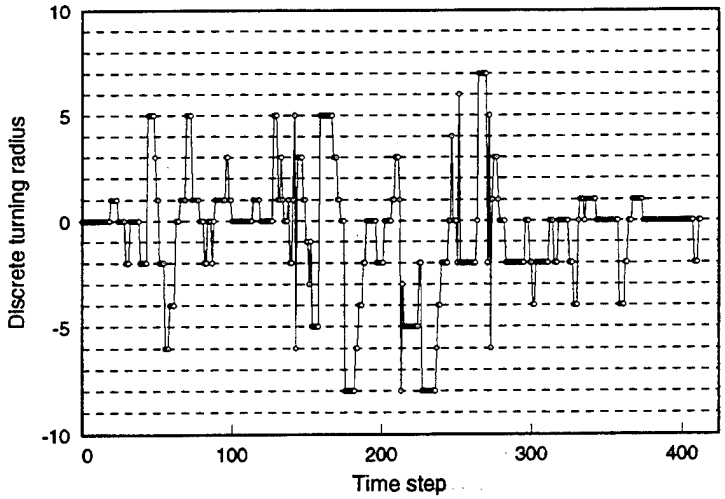


(a) Initial and goal configurations

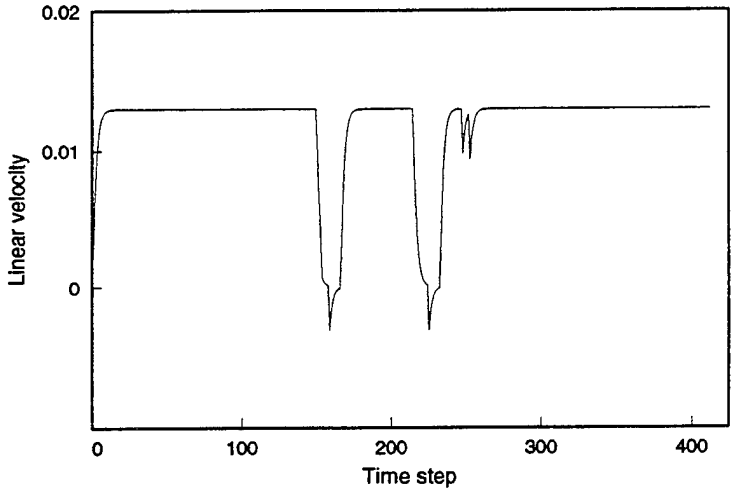


(b) Trajectory

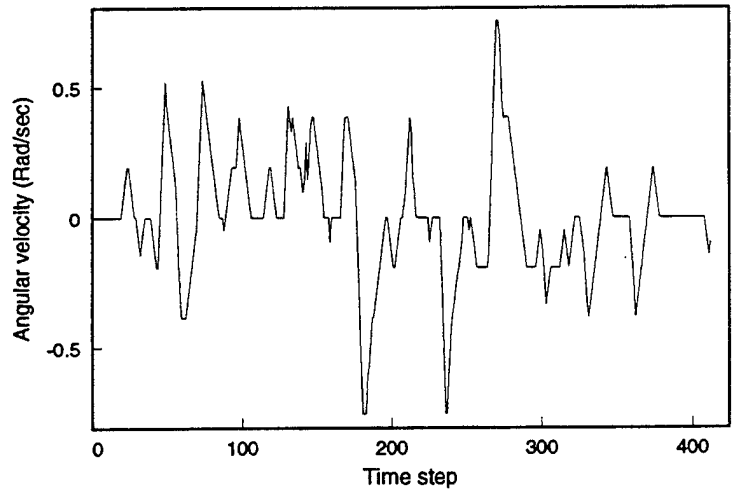
Figure C.6: Example 3: Escaping a local minimum created by unexpected obstacles



(a) Control history



(b) Linear velocity



(c) Angular velocity

Figure C.7: Example 3: Control history, the linear and angular velocity profile

Bibliography

[AF89] N. Ayache and O. D. Faugeras. Maintaining Representations of the Environment of a Mobile Robot. *IEEE Trans. Robotics Automation*, RA-5(6), 1989.

[Ark87] R. C. Arkin. Motor Schema-Based Mobile Robot Navigation. *The International Journal of Robotics Research*, December 1987.

[BL89] J. Barraquand and J.-C. Latombe. On Nonholonomic Mobile Robots and Optimal Maneuvering. In *Proc. of IEEE International Symposium on Intelligent Control 1989*, Albany, NY, USA, September 1989.

[BL91] J. Barraquand and J.-C. Latombe. Robot Motion Planning: A Distributed Representation Approach. *The International Journal of Robotics Research*, 10(6), December 1991.

[BLP85] R. A. Brooks and T. Lozano-Pérez. A Subdivision Algorithm in Configuration Space for Findpath with Rotation. *IEEE Transactions on Systems, Man and Cybernetics*, 15(2), 1985.

[Bro86] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1), April 1986.

[CCL+90] P. Caloud, W. Choi, J.-C. Latombe, C. LePape, and M. Yim. Indoor Automation with Many Mobile Robots. In *Proc. of IROS '90. IEEE International Workshop on Intelligent Robots and Systems '90. Towards a New Frontier of Applications*, Ibaraki, Japan, July 1990.

[CL85] R. Chatila and J. P. Laumond. Position Referencing and Consistent World Modeling for Mobile Robots. In *Proc. of IEEE International Conference on Robotics and Automation*, St. Louis, MO, March 1985.

[Cro89] J. L. Crowley. World Modeling and Position Estimation for a Mobile Robot Using Ultrasonic Ranging. In *Proc. of IEEE International Conference on Robotics and Automation*, New York, 1989.

[CT90] I. J. Cox and Wilfong G. T. *Autonomous Robot Vehicles.* Springer-Verlag, New York, 1990.

[CZL89] W. Choi, D. J. Zhu, and J.-C. Latombe. Contingency-Tolerant Motion Planning and Control. In *Proc. of IROS '89*, Tsukuba, Japan, 1989.

[DW87] H. F. Durrant-Whyte. *Integration, coordination, and control of multi-sensor robot systems.* Kluwer Academic Pub., Boston, 1987.

[Elf85] A. Elfes. Multiple levels of representation and problem-solving using maps from sonar data. In *Proc. of DOE/CESAR Workshop on Planning and Sensing for Autonomous Navigation*, Oak Ridge Nat. Lab., UCLA, Los Angeles, CA, August 1985.

[FB88] A. M. Flynn and R. A. Brooks. MIT Mobile Robots: What's next? In *Proc. of IEEE International Conference on Robotics and Automation*, 1988.

[Fir87] R. J. Firby. An Investigation into Reactive Planning in Complex Domains. In *Proc. of AAAI 87*, Seattle, WA, July 1987.

[Fra90] Fraichard, Th. and Laugier, C. and Líevin, G. Robot motion planning: the case of non-holonomic mobiles in a dynamic world. In *Proc. of IROS '90. IEEE International Workshop on Intelligent Robots and Systems '90. Towards a New Frontier of Applications*, Ibaraki, Japan, July 1990.

[Gir79] Giralt, G. and Sobek, R. P. and Chatila, R. A Multi-Level Planning and Navigation System for a Mobile Robot; A First Approach to HILARE. In *Proc. of 6th IJCAI*, 1979.

[Gir84] Giralt, G. and Chatila, R. and Vaisset, M. An integrated navigation and motion control system for autonomous multisensory mobile robots. In *1st Int. Symp. Robotics Research*, 1984.

[GL87] M. P. Georgeff and A. L. Lansky. Reactive Reasoning and Planning. In *Proc. of AAAI 87*, Seattle, WA, July 1987.

[Har85] S. Harmon. Knowledge Based Position Location on Mobile Robots. In *Proc. of the 11th IEEE Industrial Electronics Society Conference*, San Francisco, CA, November 1985.

[Kae86] L. P. Kaelbling. An Architecture for Intelligent Reactive Systems. Technical report, Artificial Intelligence Center, SRI International, Menlo Park, California, 1986.

[Kha86] O. Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *The International Journal of Robotics Research*, 5(1), 1986.

[Kro84] B. H. Krogh. A Generalized Potential Field Approach to Obstacle Avoidance Control. In *Robotics Research:The Next Five Years and Beyond, SME Conference*, Bethlehem, PA, 1984.

[KT86] B. H. Krogh and C. E. Thorpe. Integrated Path Planning and Dynamic Steering Control for Autonomous Vehicles. In *Proc. of IEEE ICRA*, San Francisco, 1986.

[KTB87] D. J. Kriegman, E. Triendl, and T. O. Binford. A mobile robot: Sensing, planning, and locomotion. In *Proc. of IEEE International Conference on Robotics and Automation*, Raleigh, NC, 1987.

[KTB89] D. J. Kriegman, E. Triendl, and T. O. Binford. Stereo Vision and Navigation in Buildings for Mobile Robots. *IEEE Transactions on Robotics and Automation*, 5(6), 1989.

[KV89] P. Khosla and R. Volpe. Superquadric Artificial Potentials for Obstacle Avoidance and Approach. In *Proc. of IEEE International Conference on Robotics and Automation*, April 1989.

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

[Lau87] Laumond, J. P. Finding collision-free smooth trajectories for a non-holonomic mobile robot. In *Proc. of 10th IJCAI*, pages 1120–1123, 1987.

[LK85] A. Lucas and T. Kanade. Optical Navigation by the Method of Differences. In *Proc. of 9th IJCAI*, Los Angeles, CA, August 1985.

[LL92] A. Lazanas and J.-C. Latombe. Landmark Based Navigation. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 1728–1733, Nice, France, 1992.

[LP83] T. Lozano-Pérez. Spatial Planning: A Configuration Space Approach. *IEEE Transactions on Computers*, C-32(2), 1983.

[LPW79] T. Lozano-Pérez and M. A. Wesley. An Algorithm for Planning Collision-Free Paths among Polyhedral Obstacles. *Comm. ACM*, 22(10), 1979.

[LS86] V. Lumelsky and A. Sepanov. Dynamic Path Planning for a Mobile Automaton with Limited Information of the Environment. *IEEE Transactions on Automatic Control*, AC-31(11), November 1986.

[LTJ90] J. P. Laumond, M. Taix, and P. Jacobs. A motion planner for car-like robots based on a mixed global/local approach. In *Proc. of IROS '90. IEEE International Workshop on Intelligent Robots and Systems '90. Towards a New Frontier of Applications*, Ibaraki, Japan, July 1990.

[May79] P. S. Maybeck. *Stochastic Models, Estimation, and Control.* Academic Press, New York, 1979.

[Mor77] H. P. Moravec. Towards automatic visual obstacle avoidance. In *Proc. of 5th IJCAI*, MIT, Cambridge, MA, August 1977.

[Mor83] H. P. Moravec. The Stanford Cart and CMU Rover. In *Proc. of IEEE*, volume 71, 1983.

[Nil69] N. J. Nilsson. A Mobile Automation: an Application of Artificial Intelligence Techniques. In *Proc. of 1st IJCAI*, Washington, D.C., May 1969.

[O'H73] D. A. O'Handley. Scene Analysis in Support of a Mars Rover. *Computer Graphics and Image Processing*, 2, 1973.

[Ran86] J. E. Randolph. Mars Rover 1996 Mission Concept. Technical report, Jet Propulsion Laboratory, 1986.

[Rea88] Real World Interface, Real World Interface, P.O. Box 270, Dublin, NH 03444. *Real World Interface Mobile Robot Base B12 Guide to Operations*, 1988.

[RK90] E. Rimon and D. E. Koditschek. Exact Robot Navigation in Geometrically Complicated but Topologically Simple Spaces. In *Proc. of IEEE ICRA*, Cincinnati, 1990.

[RS90] J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 1990.

[Sch87] V. Scheinman. Robotworld: A Multiple Robot Vision Guided Assembly System. In *Proc. of ISRR*, 1987.

[SE87] S. C. Shapiro and D. Eckroth. *Encyclopedia of Artificial Intelligence.* John Wiley & Sons, Inc., New York, 1987.

[Sla90] M. G. Slack. *Situationally Driven Local Navigation for Mobile Robots.* PhD thesis, Verginia Polytechnic Institute, Computer Science Department, April 1990.

[SS83] J. T. Schwartz and M. Sharir. On the Piano Mover's Problem: I. The Case of a Two-Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers. *Communications on Pure and Applied Mathematics*, 36, 1983.

[SSC88] R. Smith, M. Self, and P. Cheeseman. Estimating Uncertain Spatial Relationships in Robotics. *Uncertainty Artificial Intelligence*, 2, 1988.

[Tho77] A. M. Thompson. The Navigation System of the JPL Robot. In *Proc. of 5th IJCAI*, 1977.

[THTS88] A. M. Thompson, M. Hebert, Kanade T., and Shafer S. Vision and Navigation for the Carnegie-Mellon NAVLAB. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(3), May 1988.

[TL92] H. Takeda and J.-C. Latombe. Planning the Motions of a Mobile Robot in a Sensory Uncertainty Field. Technical report, Stanford University, Dept. of Computer Science, 1992.

[Zhu92] D. J. Zhu. *Exploring the Interaction of Geometry and Search in Path Planning.* PhD thesis, Stanford University, Computer Science Department, February 1992.

[ZL91] D. J. Zhu and J.-C. Latombe. New Heuristic Algorithms for Efficient Hierarchical Path Planning. *IEEE Tr. on Robotics and Automation*, 7(1), February 1991.