Reliably Mapping a Robot's Environment using Fast Vision and Local, but not Global, Metric Data

by

Robert S. Thau

Submitted to the Department of Brain and Cognitive Sciences
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1997

Signature of Author ....
Department of Brain and Cognitive Sciences
March 28, 1997

Certified by...
Matthew Wilson
Assistant Professor of Neurobiology
Thesis Supervisor

Accepted by ....
Gerald E. Schneider
Professor of Neuroscience
Chairman, Department Graduate Committee

Reliably Mapping a Robot's Environment using Fast Vision and Local, but not Global, Metric Data

by
Robert S. Thau

Revised version of a thesis submitted to the
Department of Brain and Cognitive Sciences
on March 28, 1997, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

A robot operating in an initially unknown environment must, in general, acquire and maintain information about that environment in order to perform tasks which involve repeated returns to particular locations, or reliable traversal of particular routes. Many practical applications of mobile robots in industrial and office environments require these capabilities. A basic prerequisite for this mapping functionality is the ability to reliably determine where the obstacles in the environment actually are.

This thesis describes a system which provides reliable mapping functionality in a mobile robot, obtaining reliable information from cheap and readily available sensors without requiring great computational resources. The basic sensor used to detect obstacles in the environment is a camera. Using assumptions about the environment which are quite reliable in common office environments, the system is able to reliably detect and estimate the range to visible obstacles at very little computational expense.

The system uses this data to build up a graph-structured representation of the environment. While the system uses local metric information, it does not assign a single set of global coordinates to any of the obstacles it detects, for two reasons. Firstly, assignment of global coordinates consistent with all observations may be impossible. For instance, if there are ramps in the environment, the measured distances between obstacles will not be consistent with any embedding into a plane. Secondly, global coordinates are not actually required for any task the robot must actually perform. However, purely *local* metric data is reliable enough to be of substantial use, and there are sometimes few alternatives to using it.

Finally, the thesis describes parallels between the internal operations of the navigation system and what is believed to be a spatial memory system operating in the brains (specifically, the hippocampi) of rodents, and discusses how more a "brain-like" system might have certain advantages over the one described here.

Thesis Supervisor: Matthew Wilson
Title: Assistant Professor of Neurobiology

Acknowledgments

Many people have helped me throughout the long course of my work here. First and foremost, thanks go to my advisers, Tommy Poggio and Matthew Wilson, for their unflagging support. Thanks also go to the other members of my committee, Ian Horswill and Peter Dayan — I would, in particular, like to thank Ian for his assistance with the robots and for assistance with project direction, Peter for his many helpful suggestions regarding presentation of the work.

This work was done in the MIT Artificial Intelligence Laboratory, a unique and convivial environment. Thanks go to the many researchers there for their stimulating discussions, and also to the staff for keeping it running (particularly Laurel Simmons and Bruce Walton).

I would also like to thank the National Science Foundation and the Office of Naval Research for providing the funding which saw me through the bulk of my time here.

Finally, of course, thanks go to my parents for their encouragement and support.

Contents

Chapter 1

Introduction

A mobile robot has to interact in a useful way with its environment. For some limited applications, it may be possible to govern this interaction entirely by immediate sensory stimuli, without having any kind of a memory. However, the behaviors which such a robot can exhibit are intrinsically limited. In particular, without strong restrictions on the environment, or some sort of external memory (as in animals' scent-marked trails) it cannot return to a "home base" or prior point of interest, and cannot advance reliably towards a goal if it has to go around obstacles to do so. Either one of these goals (or any task that would require them) can only be accomplished by using some sort of representation of the robot's environment to guide its actions — in other words, some form of map.

1.1 Long-range and short-range maps

It is worth noting that the goal we have set above (a robust representation of the robot's extended environment) is not necessarily the goal of all of the mapping papers in the mobile robot literature. Many mapping systems concentrate on building a reliable map of the obstacles present in the robot's immediate vicinity — indeed, there have been mapping systems which explicitly discarded "old" data, rather than attempting to use it to build a more comprehensive picture of the robot's extended surroundings.

The difference, to some extent, stems from a difference in the goals of these systems, which is often rather different from support for the more high-flown scenarios given above. Many robots' sensors give them, at best, an unreliable and fragmentary picture of the world around them. For instance, robots that rely on sonar receive at best range in a few discrete directions at any given instant, and even these values are typically unreliable, particularly in office environments, due to specular reflections from walls and the like.

So, a sonar-based robot has to combine sensor readings over time simply in order to form an integrated picture of the world around itself. And indeed, many robot mapping systems, particularly those designed around the notion of "Bayesian grids" or one of its intellectual descendants [33], are intended precisely to facilitate such a process; the goal is to quickly build up a reasonably accurate geometric picture of the obstacles in the immediate surroundings, using only sensor readings which each convey a limited amount of unreliable information, with less emphasis on the notion that the map should be stable over time.

At the opposite extreme are systems like those of [52], which attempt to retrieve stable data on the topology of the obstacles in the environment, but which does not attempt to recover metric data at all. The emphasis here is entirely on a map which can be used to stably guide navigation over time, and while the data recovered is indeed useful for that purpose, it is *not* useful for obstacle avoidance at all; Kriegman's system accomplishes this task by other means.

Most systems for acquiring what we are calling long-range maps (i.e., information about extended regions of space which is useful over extended periods of time) rely on, e.g., Kalman-filter based methods for estimating and tracking the positions of individual obstacles. Typically, the data structures manipulated by these systems do have a notion of the metrics of the space in which the robot lives (i.e., they are not merely topological), but rather than trying to keep track of the state of individual "cells" within a discretization of that space, they instead track obstacles as objects, retaining a considerable amount of data on each (including means and error estimates for the position of the obstacle within the larger space). (Doing all the math for the Kalman filters is quite tractable when small numbers of objects need to be considered, but as discussed in chapter 2, the computational load becomes much more significant when the amount of data which comes in from a richer sensor, such as vision, has to be dealt with).

For the purposes of this thesis, we consider all of these data structures to be maps of a sort — they are representations of information about the robot's external environment which are not necessarily available to the robot directly through its sensors. The value of such information, as we conceive it, is not in having information for its own sake, but in the tasks that having it allows the robot to perform. What we want for the robot to be able to do is to be able, at least to reliably reconstruct the paths that it took from one point to another in a relatively static environment, despite its own sensor errors; to be able to correct for reasonable errors in its own representations or in its sensors, and finally, to detect when the environment has actually changed, or it has gotten lost, and perhaps even to do something useful about it.

1.2 The need for state

A question which may be useful to pose at the outset is whether we really do need any kind of state information internal to the robot at all. Might there be some trick which allows us to dispense with it?

In fact, there are at least two tricks which have been used in robotics, both of which have precedent in biology as well. The simplest approach is just to exploit cues which have been placed in the environment to direct the robot, the classic example being the wire buried under the carpet. Obviously, that restricts the robot to paths which have been laid down in advance, but that is, in a sense, just a restriction on the implementation and not the basic idea — ants follow scent tracks laid down by themselves, or other ants, perfectly well. Humans sometimes use trail markers for similar purposes. However, it isn't always feasible to have a robot aiming a paint gun at the walls of a new environment, and use of olfactory cues is as likely to be limited by the tolerance of the humans that need to work near the robot as it is by the relatively poor state of chemical sensors in robotics, as compared to their biological counterparts.

The other alternative, which has been tried in robotic domains, is to try to construct a series of local responses to local cues which add up to globally coherent behavior which accomplishes the overall task. Indeed, there is at least one existence proof for the viability of such a scheme; the robot Herbert constructed by John Connell [12] managed to accomplish the task of finding and retrieving coke cans, returning them to a home base, by collecting local reflexes, without any notion or representation of a global plan.

The local reflexes in question were variants of the right-hand rule, which the robot would follow until its sensors had picked up a can. Once the can was retrieved, it would then turn around, reverse the rule set, and (if the environment was unaltered) return to a place which must be the same as its origin, and which could be recognized as such.

As far as achieving the task, this strategy works, but only at the cost of a separate trip for every can retrieved; what's more, the robot is confined to recapitulating all of its steps, and is incapable of recognizing, for instance, that a given area has already been cleaned and it should next try somewhere else. Secondly, if the environment were ever to change, the robot would find itself instantly, and irrevocably, lost.

(Nevertheless, it is worth noting that there are entomological parallels here too, in particular to the well-known behavior of certain wasps which appear to manipulate their eggs and their environment in an extremely purposive way, which turns out to be nothing more than a long string of wired-in reflexes. However, such long strings of reflexes are most often seen in circumstances where the wasp is likely to encounter an environment which is stable enough that the wasp itself is the only active agent — meaning a set of actions extending over not too many minutes in a tightly circumscribed areas. Or to put it another way, the way these strategies are actually used by insects, it is in such a way that they are unlikely to encounter the bugs. As a procedure for navigation, it is unreliable; insects which accomplish such tasks reliably tend to use laid-down trials (ants), displacements to landmarks (bees), or otherwise more sophisticated mechanisms).

1.3 The need for acquisition

So, our robots need maps. Where are they to come from? One alternative would be to program them in manually. There are two disadvantages with such an approach. The most obvious is that manually building a useful map of a large area is expensive. However, it is also true that without a way of checking and updating its map, a robot has no way of dealing with changes in its environment, something which may be important in some applications.

Animals face the same difficulties, of course — the "wired in" solution is simply not available, to, say, a rodent in an urban environment (which may be constantly changing, as dumpsters move and apartment dwellers jam up passages with steel wool), or for that matter any animal which has to forage in an environment where the available food supplies are reliable only for a limited amount of time (fruit trees, watering holes).

Animals also have effective solutions to these problems. Examining what is known about these strategies, and comparing them to robot mechanisms, may yield parallels of interest. So, it is worth noting some aspects of the animal solutions, and their known neural correlates.

1.4 Neural correlates of navigation in rodents

The most widely studied neural correlate of navigational information is the "place cell" phenomenon in rodents, particularly in the rat and (more recently) the mouse. these cells have activity patterns which appear strongly correlated to navigational aspects of the animal's behavior — specifically, if the animal is allowed to explore a given environment, some cells in the hippocampi of these animals have been shown to develop "place fields" — that is, they fire when the animal is in a particular place, and not when it is elsewhere. (As discussed in chapter 6, the activity does not appear to be directly responsive to learning of the cluster of stimuli which prevails at a particular location — for instance, if the primary cues to location in a given environment are visual, and the lights are turned out, place cell activity is not extinguished).

There are parallels between the neural structures revealed by this activity, and the data structures built and used by our algorithms. There are, however, differences as well — which may reflect the underlying differences between the structures of brains and those of computers. See chapter 7 for further discussion.

1.5 Goals of this work

To return to where we started, for a mobile robot, learning about and reliably navigating in an initially unknown environment is clearly a problem. Our goal is to provide solutions to that problem which are reliable, robust, and cheap.

Vision is the primary visual cue used by humans; this serves as a basic existence proof that it is possible to do useful navigation based solely on vision. We aim here to do so cheaply, and robustly. The minimal visually guided robot is a single camera, an omnidirectional robot base, and a small computer to provide intelligence, and we have attempted to achieve robust performance using no more than this. Note that a camera and the associated hardware required to grab images into RAM can be had quite cheaply; the basic short-range mapping algorithm described in chapter 3 have run on systems costing a few hundred dollars for all electronics, including the computer.

In fact, our long-range maps are built out of a successsion of short-range maps, built up quickly and cheaply through use of visual strategies tailored to the environment (following Horswill [21]). The maps are built up into a graph structure which represents local metric information without imposing any global coordinate system; this allows for proper function in situations where the metric information available to the robot are not, in fact, globally consistent.

The strategies we adopt have intersting parallels to some of what is known about the biological underpinning of navigation in rodents. These are explored in chapter 6.

1.6 Overview of the final system

Finally, I conclude this introductory chapter with a description of the system which we finally build — remaining chapters will help to motivate the specific features, but in this example, we seek to simply give an overview of the system we have built, and how it supports our overarching goals, to orient the reader through future discussion.

1.6.1 Short-range mapping

As indicated above, the maps that this system builds are long-range maps built on a succession of individual short-range mapping outputs. The short-range mapping technology is therefore basic to the results achieved with the resulting system.

The short-range mapping technology presented in this thesis is fast, robust, and fairly cheap (the only sensing required on the robot is one camera). It is based on the assumptions that drove the obstacle avoidance algorithm from the work of Horswill [21], which is one of the more robust obstacle-avoidance algorithms in existence for the environments which it is meant for.

Specifically, the algorithm requires a clear, relatively flat floor which is visually distinguishable from the obstacles. Given this basis (very common to office environments, for example), it performs obstacle avoidance quite robustly.

By combining the raw data from the visual system which is needed to support the above obstacle-avoidance algorithm (a single camera), with a few easily recovered parameters of the system's geometry, we are able to develop a simple and very robust approach to building short-range maps in these environments. In effect, this approach uses the camera as a fairly cheap, and very accurate, high-grade range sensor.

Considered simply as snapshots the maps are very accurate. These snapshots can be used to crudely build larger maps by overlaying the snapshots of the obstacles seen in different positions, displaced by the distances indicated by the robot's own dead reckoning. A map built using this technique is shown in figure 1-1; this traverse involved repeated traversal of the same path in a relatively large space, with furniture and other obstacles in the middle.

Indeed, the major source of noise in the resulting maps is not in the sensing, but rather in the flaws in the robot's dead reckoning, particularly in carpeted environments. This can be seen in figure 1-1 in the way that multiple traverses of the same space, in the lower left corner of the map, give rise to multiple representations of the obstacles, slightly shifted with respect to each other.

Figure 1-1: A short-range map of a large environment

In fact, this experimental data allows us to visualize directly the amount of noise which a full mapping system has to deal with, and for which the long-range component has to compensate. The displacement over the ten minutes or so of continuous operation represented above amounts to several feet. This drift is large enough to require some form of effective compensation. However, it is small enough to strongly limit estimates of the robot's actual positions relative to some prior map, given a prior position fix. We therefore seek to build more elaborate representations, with short-range maps of this sort as a primitive.

1.6.2 The complete system — a simple traverse

As indicated above, we build from this to a long-range solution by linking the short-range maps into graphs. An overview of the process:

We begin with the robot exploring a simple simulated environment, as depicted in figure 1-2. The conditions of the simulation included substantial amounts of simulated noise both in response to robot base motion commands and in the degree of slip between those and the sensed (dead-reckoned) position that resulted. That is, not only was the response to a motor command randomly disturbed from what was actually commanded, but the perception of the actual effect had additional noise added beyond that, both in terms of pure positional error, and error in relative heading. (The latter error mirrors the consequences of precession in the RWI robot bases, which is a very common cause of error if the wheels on the robot base are slightly misaligned).

The robot begins in the middle left. It proceeds around the large loop in the environment. As it does so, it makes a series of *place representations* which are formed into a graph, with links denoting neighborhood relationships.

The robot proceeds around the long corridor, eventually going full circle and arriving at a place it had previously occupied. It must at this point (see figure 1-2) recognize that it has reached a place it has already occupied, despite possible drift in the robot's estimates of both heading and absolute position.

Having done this, it must then proceed to explore some other section of the environment. To do so, it must get from the place it currently occupies (one whose exits have all been completely explored) to another place, which has unexplored exits. Now, we have built a map whose purpose is precisely to support such traversals, so the obvious thing to do is to use it.

Specifically, if we have the robot note, when leaving a place, whether it has unexplored exits, then in this sort of situation, where the robot has completely explored the space immediately surrounding the location it currently occupies, it can choose the closest of the other places which have been noted as incompletely explored, lay out a path to it, and seek to that place, subsequently resuming exploration. This is shown in figure 1-5, which shows the robot returning to the opposite corner and beginning again to explore.

Finally, the environment, being finite, is tapped out. The complete representation, as it is finally built, is shown in figure 1-6.

This is a basic outline of the system we describe in the remainder of this thesis.

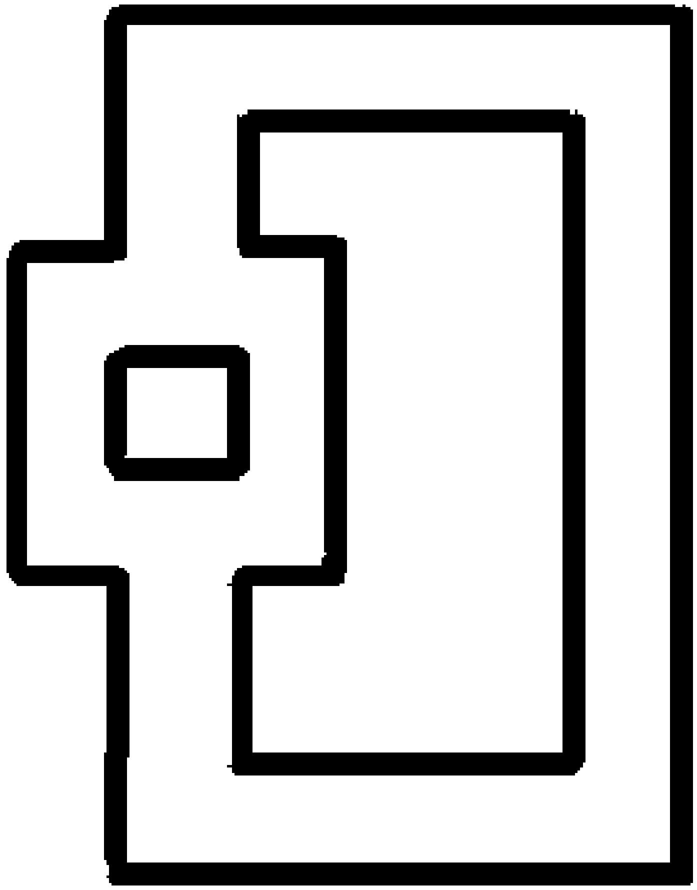


Figure 1-2: Environment for sample, simple traverse

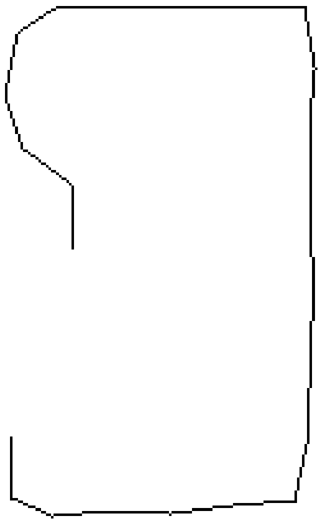


Figure 1-3: Path followed by the robot, shown by a representation of the (very simple) graph which is built to represent its traverse)

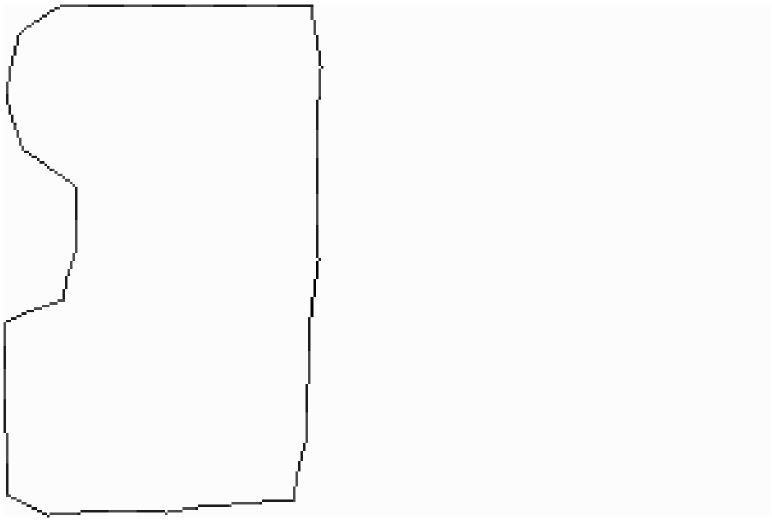


Figure 1-4: Path followed, and the graph, as the traverse continues. The robot has gone in a large circle, and the graph must be made to reflect that.

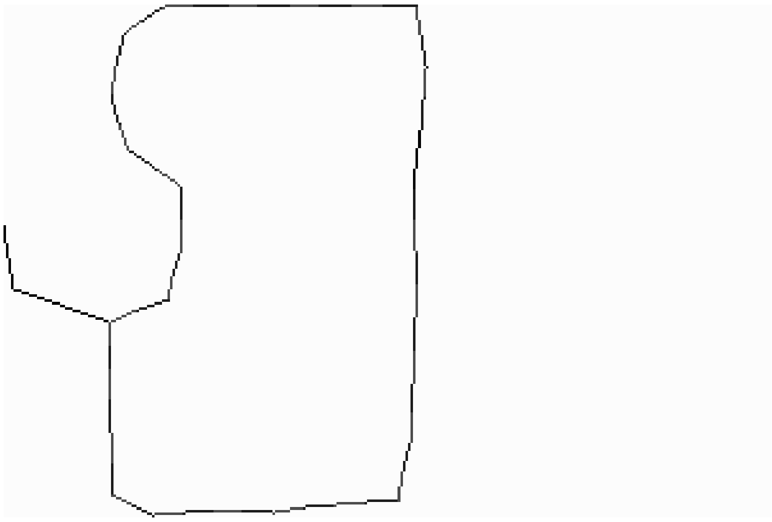


Figure 1-5: Path followed, and the graph, as the traverse continues further. The robot has retraced its steps to get to a location which it had not yet completely explored.

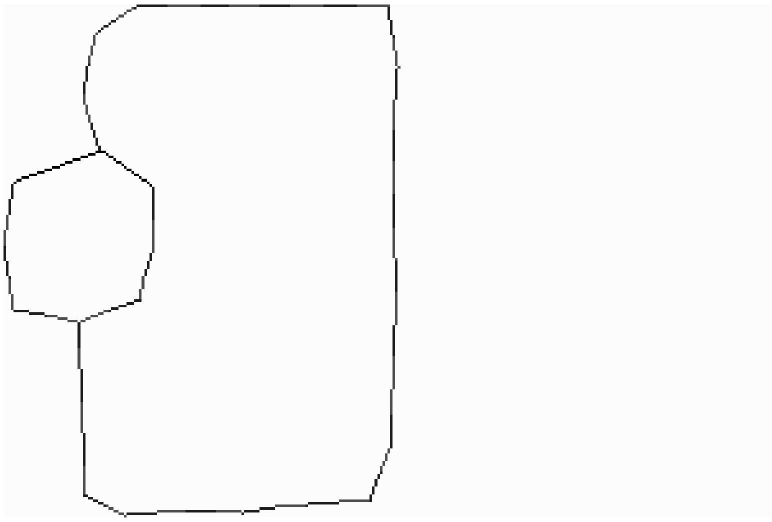


Figure 1-6: Final representation of the sample environment.

It illustrates the sytem's basic capabilities, and some of the means that it uses to achieve them. It does not illustrate all the system's capabilities — in particular, it does not illustrate how the robot deals with situations in which it appears that the external environment has changed, and the map must be adjusted to compensate. (For these, see chapter 5).

1.7 Relation to biology

One very interesting question is whether there is anything about the representations bulit by this system which relates to the biological underpinnings of the solutions to the same problems in animals, which encounter these problems daily as they traverse their environments.

The apparent biological basis of this sort of behavior is best studied in rodents, where cells have been found in the hippocampi whose patterns of output appear to be strongly related to navigational functions. Specifically, as discussed in chapter 6, when the animal spends time exploring an environment, these cells develop "place fields", in which the cell fires when the animal occupies a particular place. Some properties of these representations are strikingly reminiscent of our place representations; others differ, in manners which may reflect the unique properties of the brain as a computational structure.

1.8 Overview of the thesis

The remainder of the thesis reviews the background of the navigational problems discussed above, and describes our approach to those problems. Specifically chapter 2 reviews mapping, and chapter 3 describes our solution to short-range mapping. Chapter 4 discusses what data we extract from the short-range maps for long-range use —

what information is extracted into representations of a particular locale, and how we match that information to a subsequent short-range map which may or may not come from the same locale. Chapter 5 discusses how we assemble these "place representations" into a comprehensive picture of the environment. Chapters 6 and 7 explore the relations to biological systems, and chapter 8 gives conclusions and discusses further work.

We begin with the review.

Chapter 2

Previous work

2.1 Introduction

There is a wide variety of mapping systems for robots in the literature. These vary greatly in their approach, for several reasons. These include, of course, the capabilities of the robot itself, but the differences go beyond that. In particular, as discussed in chapter 1, the basic purposes of robot mapping systems differ substantially — ranging from "scratch pads" noting the locations of obstacles which may not be reliably detected from all angles (see section 2.3.1), to systems which attempt to build persistent, reliable and robust representations which allow the robot to get around the environment, but may not note the exact or even relative positions of any obstacle at all, collecting only topological data regarding paths and connections.

This chapter surveys a number of mapping systems from the literature. In general, it proceeds from less abstract representations of the robot's environment (e.g., grids which directly represent the contents of the robot's workspace), to more abstract ones (e.g., the purely topological maps). For brevity's sake, it concentrates mainly on systems in which the robot does not begin with a preloaded or precomputed "picture" of its environment, and is required to build, or to learn, the map on its own (though it briefly discusses systems a few systems in which the map is "programmed in" where discussion of those systems helps to elucidate their learning counterparts). One unfortunate consequence of this is that the main flow of discussion skips over the system of Horswill [21], whose obstacle-avoidance technology is fundamental to the short-range mapping strategy discussed in chapter 3; however, the necessary details will be covered there.

As the scratchpad, or "short-range", mapping systems discussed in chapter 1, which use the map mainly to better deal with their immediate environment (as opposed to, say, using them for long-term planning), tend to be most explicit and least abstract in the data that they compute and store about the environment, the discussion will tend from "short-range" to "long-range" systems as well. We begin with the most direct approach, which is commonly used only in short-range systems.

2.2 Direct representation of the environment

If we assume that a robot's environment is roughly planar (an assumption which may not be valid in the presence of hills in outdoor environments, or ramps in indoor ones — a point that we shall, eventually, return to), then the most direct way of representing the contents of that environment in the memory of a computer acting as a controller of the robot is to simply create an array $\mathbf{map}(i,j)$ in which the (i,j)'th cell contains information about the immediate neighborhood of the point (i,j) in the environment (for some suitable scaling of units to array indices, which is governed by practical considerations including the robot's own physical size and general rate of travel). Many mapping systems are indeed based on exactly this sort of a mapping strategy — that is, the creation of a grid-based data structure which contains some sort of information at each cell regarding the nature of a small neighborhood of the environment.

This chapter discusses a set of simple approaches which have proven themselves suitable for many applications, based on directly representing the space through which the robot travels (sometimes referred to as the workspace). As we shall see, these methods are robust when the robot is confined to a relatively small area, but they may run into problems with large workspaces, for several reasons, including difficulties in dealing with sensor drift, and the memory required for the representation itself (in most cases proportional to the area — or, for underwater mobile robots, the volume — of the space occupied by the robot).

There is a wide variety of systems along these lines in the literature, but there are certain common themes in their operation. The common outline is roughly as follows:

2.2.1 Grid-based mapping — an outline

We begin by assuming that we are building a map within a mobile robot. The robot has effectors which are capable of returning some odometric indication of how far the robot has traveled (and how far it has turned) as time progresses; these may be used to compute a dead-reckoned estimate of the robot's position (though that will, of course, be subject to error, due to such causes as wheel slip, sensor noise, wheel misalignment, and so forth). In addition, the robot is equipped with one or more sensors capable of detecting objects at a distance — SONAR and laser range-finders being among the more common types of sensors in use.

The robot, so equipped, travels through its environment, maintaining an estimate of its own current position, and making periodic observations. Each observation develops evidence as to the contents of certain specific neighborhoods in the robot's immediate vicinity; as this evidence comes in, the robot updates the corresponding cells in its grid-based map. The map may then be used to determine the direction of any further travel. For instance, the robot may change directions in order to avoid an area which contains grid cells indicating that obstacles are present (or perhaps that they are likely to be present).

If all sensors are presumed to be reliable, then a mapping system along the lines given above is simple to construct and reliable as well. Unfortunately, this is frequently not the case. All sensors have noise, and that must be taken into account when updating the grid, so that spurious readings to not lead to the creation of spurious map features. As we shall see, this problem is particularly severe with SONAR

— an inexpensive and widely used sensing method, but one subject to a wide variety of errors.

2.2.2 Grid-based mapping — the Bayesian approach

To summarize the basic grid approach outlined above, a robot using this technique will travel through its environment making periodic observations. At each observation, it will update grid cells about which it receives new information — that is, information regarding the likely presence (or absence) of an obstacle in the small neighborhood of space described by that grid cell.

Two questions arise in the design of such a system. First, the we must determine what sort of information the robot stores in the grid. Since no single SONAR pulse can convey definitive information about any single point in the environment, this data structure must allow for the representation of partial information, and for the accumulation of (possibly conflicting evidence) about what lies in a given area over time. Second, the robot must determine what real-world areas it can draw new conclusions about based on the return time-of-flight seen for any given SONAR ping, and hence which cells of the grid to alter (in whatever manner is appropriate to its representations) based on that observation.

The most common general technique is to use some sort of probabilistic technique to deal with the uncertainty. This technique was popularized by Hans Moravec [33], though other sources trace it farther back to earlier work. Since Moravec's technique has been widely copied and extended, it deserves at least a brief explanation.

In Moravecs' maps, each grid cell on the map contains a single real number, which is seen as either the probability of there being an obstacle in the corresponding region of the workspace, or some simple numerical function of that probability, such as the natural log of it. (Storing logs of probabilities, instead of storing probabilities directly, may be useful in saving a little computation — for instance, by converting multipliciations into additions — in the updates, as combination of probabilities frequently does involve multiplicative operations).

Areas with a very low probability of containing an obstacle are considered "safe" for the robot to travel into; areas with a high probability of containing an obstacle are considered "dangerous" and avoided. Areas with an intermediate value cannot be considered safe for travel, but may be considered candidates for further exploration.

Now, what the designers of a robot begin by knowing about any given sensor is the probability of a given sensor return (say, a particular time-of-flight return for a given SONAR ping) given the presence of an obstacle at a particular point. However, what the robot needs to know, given that it has in fact received a particular sensor return (and none other), is what that tells it about the probabilities of obstacles at various points. Bayes' rule allows the robot to determine one set of probabilities given the other, and given some notion of the probabilities of obstacles being there. That is, the probability of an obstacle existing at point **x**, given a particular sensor return **r** can be determined by:

$$\mathrm{P}(x|r) = \frac{\mathrm{P}(r|x)\mathrm{P}(x)}{\mathrm{P}(r)}$$

Note that this requires some notion of the *prior probability*, $P(\mathbf{x})$, of there being an obstacle at point $\mathbf{x}$, before the observation has been taken. Moravecs' robots read these prior probabilities right out of the map, and then write updated probabilities for all grid cells affected by the observation back in. Of course, this cannot apply to the initial conditions of a truly unknown environment, in which the best option available is simply to initialize the grid to a uniform value denoting uncertainty. (Note that, in any case, this procedure amounts to treating the results of successive observations as independant, which is not, strictly speaking, entirely true).

2.2.3 Grid-based mapping — other approaches

It is worth noting that there are other possible approaches to management of a grid. For instance, Oriolo et al. [38] manage the uncertainty by using multiple fuzzy sets, as opposed to using a Bayesian framework. That is, the robot maintains two fuzzy sets of cells (that is, for each cell, it maintains two "probabilities of membership", one for each set). The two sets are the set of cells probably containing an obstacle, and the set of cells probably containing free space. It may be misleading to ascribe the difference between these two systems to the use of fuzzy sets, per se, as fuzzy set membership functions can be viewed as probabilities themselves, and from that perspective, the use of a *single* fuzzy set would not be different in any substantive way from the use of a simple Bayesian grid as outlined by Moravec. However, the use of two sets (or, equivalently, the association of two variables with each grid cell) does allow this system to distinguish between cells about which there is ambiguous information (and hence, nonzero membership probabilities for both sets), and cells about which there is no information — a useful distinction in some circumstances, which is unavailable in Bayesian grid systems which use probabilities in the middle range (around 0.5) to denote both conditions.

For yet another approach, Weigl et al. [56] use a somewhat ad hoc procedure involving directly summing "obstacle hits" into the grid, and then normalizing each grid cell with respect to its neighbors. The idea here is that genuine data points will be reinforced, and spurious ones will tend to be suppressed by the normalization. (What is key to the success of this method is that only *borders* of actual obstacles show up on the grid; if this were not so, the method would fail trivially in cases of obstacles with a physically nontrivial extent).

2.2.4 Grid-based mapping — a note on applications

It is worth noting explicitly that while the above discussion has dealt with the question of errors in the detected positions of obstacles, it has not dealt with the question of errors in the robot's estimate of its own position or heading, which will of course, also affect the positions which it assigns to new obstacles as they are placed in the map. This consideration has been omitted from the discussion because a lot of robot mapping systems do not, in fact, attempt to compensate for positioning errors in any systematic way.

Obviously, in a system which is intended to produce a map for long-term usage, allowing the robot to continually maintain an estimate of its own position, plan routes, and so forth, this is a serious drawback. However, that is not actually the purpose of all systems in the robotics literature — or, in fact, many of the systems discussed in this chapter.

Rather, as we shall see, the purpose of many mapping systems is simply to allow the robot to deal more effectively with the environment in which it immediately finds itself — whether or not this results in producing a data structure that is of future use once the robot leaves, say, the room or hallway that it happens to be in.

2.3 Sensors

As discussed above, sensors commonly used on real-world robots are unreliable enough that a robot which wants to use them to build up an accurate picture of its environment is essentially forced to combine multiple observations in order to do so. Since a lot of robot mapping systems in the literature are shaped by the properties and limitations of their sensor devices, it is appropriate to consider those now, briefly.

2.3.1 SONAR

The most widely used sensor in mobile robotic experiments, at least for indoor robots, is SONAR — an acronym for SOund Navigation And Ranging. Specifically, robots tend to use active SONAR, in which the robot emits a sensor chirp at a specific target frequency, and then senses the time at which it hears an echo, and the strength of the echoes received (which may be used to deduce the approximate range of the remote object giving rise to the echo). Use of SONAR for mobile robotic experimentation has, in particular, been abetted enormously by the availability of the relatively cheap SONAR emitter/detectors originally developed by Polaroid for use in its SX-70 camera line; indeed, it is common to see robots with rings of Polaroid SONAR devices around them, allowing the robot to take range readings in several directions simultaneously without having to rotate.

Ideally, a SONAR device would emit a perfectly directional sound beam, which would be reflected directly and completely back to the robot, allowing a fairly precise estimate of the range and distance of the object giving rise to the echo. (Indeed, one occasionally sees simulations in which SONAR devices are incorrectly presumed to behave in this delightful fashion). Reality, regrettably, is somewhat more complicated, for a few distinct reasons.

To begin with, the sound beam emitted by the SONAR device is not perfectly directional — it spreads over a fairly wide angle (commonly estimated as about 20 degrees for the Polaroid device). By itself, this gives rise to some considerable uncertainty in the origin of an echo from a relatively far-off location.

Worse, the echo of the (still somewhat directional) chirp may not return directly to the SONAR, but may bounce somewhere else first. This is particularly a problem in environments containing such artificial features as completely flat walls, which may well behave as largely specular reflectors for sound — which is to say that most of the energy of a sound pulse which hits the wall at an angle will "bounce off" in the same manner that a rubber ball would if thrown at the wall, rather than returning directly in the direction in which it came. Depending on how the pulse "bounces" after that, the originating SONAR on the robot may see a delayed return (after one or more subsequent reflections), or none at all.

The end effect is that flat planar features such as walls and file cabinets are reliably detected by SONAR only when the direction of the SONAR pulse is normal to the plane of the feature, or nearly so — otherwise, it bounces off, yielding either a late return or none at all. (There are still features, even in a typical office environment, which are reliably detected by SONAR at any angle; these include corner reflectors made up of multiple planar features, and the occasional material which reflects incoming sound diffusely in all directions. However, the existence of these reliably detected features cannot mitigate the *failure* to reliably detect a class of obstacles — e.g., walls — in which the robot has a definite interest, in order to, e.g., avoid attempting to drive right through them).

Now, this doesn't mean that SONAR is useless. The information gained from taking SONAR observations at different bearings at a given location will still contain valuable information about the obstacles in the robot's immediate vicinity. However, that information may well not be complete, and the relationship between the time-of-flight of a SONAR pulse and the distance and bearing to the object giving rise to it is likely to be quite a bit more complicated than we might ideally wish. Also, while observations at a given location are likely to be stable, moving slightly may have somewhat surprising effects (if the small motion causes pulses in a particular direction to take a different "bad bounce").

In short, a SONAR-equipped robot cannot count on getting reliable and accurate readings on the location of obstacles based solely on sensor readings from one place in the environment. Combining readings from several locations is *necessary* to produce an map accurate enough to be useful for any purpose, including even short-term obstacle avoidance. (For this purpose the map may be thrown away fairly quickly — for instance, when the robot leaves a room it formerly occupied, but it must be built anyway, since otherwise, the robot will simply fail to avoid large classes of common obstacles, such as a wall approached at an oblique angle).

Indeed, many map-building SONAR robots use the map solely as a tool for obstacle avoidance, a purpose for which long-term stability of the map is not needed (and may even be a detriment, if it interferes with the robot's ability to adapt to transient obstacles); that is, in terms of our taxonomy, for strictly short-range purposes.

SONAR grid mappers tend to vary along several dimensions. In addition to variations in the kind of information stored in the map, as discussed above, and in strategies for dealing with uncertainty in the robot's position, discussed in section 2.5, another major point of differentiation is the approach taken to modeling the sensors themselves.

One approach is to simply limit the robot to intercepting features detected reliably from all angles (such as corner reflectors); the systems described by Leonard and Durrant-Whyte [27], and Leonard, Durrant-Whyte and Cox [28] take this strategy (some details of which are discussed in section 2.5 below). Unfortunately, such maps are necessarily sparse, at least in common office environments, and so not useful for immediate short-range obstacle avoidance.

Another approach is to attempt the best model possible of the vicissitudes of the SONAR sensor one possesses, in order to give a probability-based grid management strategy such as those outlined above the best chance possible. Approaches to this vary from, at one extreme, Moravec's original work [33], which attempts to reason from first principles, to Oore [37]'s approach, which works from detailed modeling of the properties of the particular sensors on his robot as measured in his environment (with some risk of overlearning environmental peculiarities).

Finally, there is a considerable literature devoted to building better SONARs — for instance, by adding a second sonar receiver, so that differential time-of-flight can be used to determine the direction from which the SONAR pulse returned (as in Stevens, Stevens, and Durrant-Whyte [48]); however, such work generally focuses on engineering considerations which are beyond the scope of this review.

2.3.2 Laser range-finders

Another, somewhat less widely used sensor, is the laser range finder. This device emits laser pulses into the environment, and, by means of interferometry, determines the distance to the object from which any response on the appropriate wavelength is seen. The advantages of this technique over SONAR are many — the direction of the beam is more precisely known, eliminating uncertainty in position due to beam spread; pure specular light reflectors (i.e., good mirrors) are much rarer in office environments than pure specular sound reflectors, making another major source of SONAR errors rarer for the laser as well. The major disadvantages are cost, and the environmental impact of even low-power lasers being beamed all over the robot's workspace (which may also be occupied by people, or for that matter, by other light-sensitive mechanical equipment).

One useful characteristic of laser range-finder data, aside from its improved accuracy over SONAR, is the density of the data received from the LRF. Since light moves much faster than sound, it is possible to take multiple observations in a "sweep" of the laser beam across the environment, without having the observations interfere with each other, in a manner that would not be possible with SONAR. Consequently, a robot with an LRF *can* produce a detailed and fairly accurate grid map of its immediate environment based on a fairly rapid series of observations from a single location — something which is not possible for a purely SONAR-based robot.

Once again, a lot of the interest in this sort of mapping technology is for use in determining not position in a large environment, for route following, but for the rather simpler and more mundane tasks of assuring that basic moment-to-moment navigation does not lead the robot into dangerous situations. So, for instance, the Ambler planetary rover [23, 24] (a very large six-legged walking machine) constructs a detailed map of its immediate surroundings, mainly for the purpose of being sure to place footfalls in such a manner as to keep the robot supported — for instance, by *not* stepping onto a rock which might roll, or a surface so steep that its feet are likely to skid off.

Another mapping system along these lines is that of Stentz and Hebert [47], for an outdoor vehicle (this one based on a HMMWV chassis, with a laser rangefinder mounted above the cab and computers in the back). In this case, the system is given the goal of navigating to a point at a particular range and bearing from its present location. That is, once again, the goal of the mapper is less to build up a picture of the environment to improve further performance wthin the same environment, than it is to simply correlate information about the robot's immediate environment, for nearly immediate use. As a consequence of this, the system once again does not attempt to compensate for odometric error in building its map.

(Note that in the outdoor vehicle arena, this is not necessarily a handicap, in the present state of the art. Through such means as GPS, an outdoor vehicle can easily obtain information about its own present position within a hundred meters; and advanced techniques such as differential GPS permit knowledge of position within a fairly large area to a much better resolution than that. Accordingly, use of other means to obtain at least a coarse estimate of position — and for fine estmates, it may well be adequate to use heuristics such as being willing to guess that if you are on an east-west road, and there is only one such road within a hundred meters of your GPS position estimate, then that is the road you are on).

This is not to say that all use of laser range-finders is for these sorts of purposes; Asada [1] describes a system which translates range readings to heights of obstacle points at a given range, and which stores those heights on a grid, using the resulting information for path-planning purposes in the usual wheeled-robot manner (treating heights within a certain range as "floor").

2.4 Making grids stable

The mapping systems described in the previous section are generally fairly simple and straightforward (though such complicated variants as that of Oore [37] do tend to stretch the point); we have demonstrated a system based on this notion which uses visual input to produce a highly reliable map of the obstacles surrounding the robot and immediately visible to it. We have also described how, by stitching such maps together, the robot may be able to build maps of an entire area, over a brief interval of time, with fair reliability. This system has been tested in an actual robot, and over short durations, it works rather well.

However, as we have seen, over longer intervals of time, when the system is deployed in a real robot, problems emerge. These emerge, specifically, from the accumulated error in the robot's own position estimates, leading, in turn, to errors in the placement of obstacles in the map (these being, of course, above and beyond those errors originating in the sensing procedure itself).

From this point forward, this chapter discusses the nature of these sorts of problems, and mapping methods in the literature which attempt to deal with them in various ways. We begin by discussing the problems themselves in somewhat more detail, then go on to consider a variety of different approaches. Some explicitly model the space being mapped and the positions of obstacles within it (typically as a euclidean 2-d plane), maintaining information about the placement of discrete obstacles on that plane (including, in some cases, error estimates) so that conflicting position data, and the like, can be reasoned about explicitly. Others, maintain only information about the structure of the environment (in extreme cases, topological data only), without storing metric information. In addition, there are different learning strategies to go with the different data structures.

We begin, however, by describing in more detail the problems we have observed with using the sorts of algorithms described above over extended periods of time, and extended traverses of distance.

2.5 Why not just a short-range map?

To begin with, let's study some of the attempts to apply the various grid-based methods covered in the previous chapters to construction of a long-range map — that is a map (or representation of the robot's environment) which is stable in the face of environmental drift.

To begin with, consider the case that we do nothing special about the problem. Now, suppose the robot travels by some route through the environment, and (perhaps by retracing its steps, perhaps not) finds itself at the same place it began from. However, due to accumulated drift in the odometry, it will actually believe it has returned to a position displaced by some amount from its initial position. Therefore, the positions of all obstacles entered on the grid will also differ from the positions at which those same obstacles were entered initially — thus giving rise to the sort of double and triple images seen on the actual grid map of figure 3-3.

Clearly, this sort of behavior has undesirable consequences for long-term operation. In particular, we might wish — and indeed, expect — that when the robot transits through the same area repeatedly, its knowledge of that area should perhaps improve and should certainly not be degraded. However, given the behavior described above, repeated traverses over the same area can not only fail to improve the robot's knowledge of that area, but can actually make it worse.

To summarize, noise in accumulated place estimates may lead to errors in the estimated relative position of observed environmental obstacles. If these are small, they can be fudged, but if the become large enough — which is partly a function of environmental factors which the robot can do nothing about — we need some approach which deals with it explicitly.

2.6 Local and Global grids

Given the success of grid-based methods over the short run, it is tempting to try to rescue the approach somehow. For instance, we may begin by noting that a grid constructed over short ranges is generally fairly reliable. In particular, a grid is quite reliable if it integrates only those particular sensor readings which are observed while the robot is at a fixed position. (Depending on the sensor structure of the robot it may be entirely sensible to use grids for such an application — for instance, recall that grids are commonly used to deal with the ambiguities of SONAR returns on real robots in environments which include specular sound reflectors such as walls).

This suggests the following strategy: build local maps from whatever sensory data are available at the robot's current position, or over a short traverse, and then merge these maps into a larger grid after applying some sort of position correction, perhaps derived from an attempt to match the data in the "global" grid to the data in the "local" grid in order to determine the best fit to what is already there.

For example, Schiele and Crowley [43] take just such an approach. Their SONAR-based robot maintains a "global" grid, which in effect maintains a picture of the environment as the robot believes it to be (including physical positions at which obstacles have been detected). The robot, of course, also maintains and updates a notion of its position within the environment (by coordinates used to index the grid). It also maintains an estimate of the uncertainty in its current position, which will grow over time due to navigational error, but may shrink if the environmental features it currently observes are a good match to the features near the current regionof the global grid. In this case, of course, the robot will correct its position estimate as well to take into account the new data.

At each timestep, it proceeds by first building up a local grid, then using it to adjust its estimate of the robot's position with respect to the coordinate system embodied in the global grid. Finally, it updates the global grid with data from the local grid (after post-processing it in order to take into account the uncertainty in the robot's position).

Maintenance of a position estimate and the associated uncertainty (in the form of a covariance matrix among the components of its state vector) are done in this system by means of a procedure known as the Kalman filter. As this same procedure is also used, in different ways, in a variety of other navigational systems, it is useful to describe it here in some detail.

2.6.1 Overview of the Kalman filter

The Kalman filter originates in the field of signal processing (hence the "filter" name, on analogy with other procedures for, e.g., modeling (and hopefully removing) noise on a communications line). However, unlike most actual filters, it is, in fact, a discrete procedure — that is, it models the evolution of a system which evolves through discrete time steps.

Specifically, the Kalman filter treats a system which is governed by the equation

$$\mathbf{x}_{k+1} = \mathbf{T}\mathbf{x}_k + \mathbf{v}_k$$

where

- $\mathbf{x}_k$ is the *state vector*, at timestep k, consisting of a number of real variables describing the state of the system (e.g., position, orientation)
- $\mathbf{T}$ is the *transition matrix*, which governs the evolution of the state vector (save for the influence of the noise sources).

- $\mathbf{v}_k$ is a set of *noise processes*, assumed to be zero-mean Gaussian, which also influence the evolution of the system (so that $\mathbf{w}_k^j$ is the sample of that process at time step k).

The Kalman filter does not assume direct access to the state vector, or noisy measurements thereof. Rather, it assumes that the environment periodically makes available observations of portions of the state, which are themselves linear transforms of the actual state vector, and may be corrupted with additive Gaussian noise of their own:

$$\mathbf{z}_k^j = \mathbf{H}^j \mathbf{x}_k + \mathbf{w}_k^j$$

where $\mathbf{z}^j$ is the j'th possible type of observation, the matrix $\mathbf{H}^j$ governs the relationship between the state vector and that observation, and $\mathbf{w}^j$ is the associated Gaussian noise process.

What the Kalman filter does is to maintain two items: an *estimated state vector*, and a covariance matrix which indicates uncertainty in the knowledge of that state vector, by predicting the evolution of the system, using the transition matrix, and then adding in corrections gathered from whatever observations are available. The magnitude of the corrections is determined, of course, by the covariance matrices — observations with high covariance have relatively little influence on the estimated state vector, or its covariance matrix, while observations of low covariance will transfer that low covariance into the relevant components of the state vector pretty much directly.

The computation required per update is some nontrivial matrix algebra — e.g., inversion of an $n \times n$ matrix, for a system with an order n state vector to be estimated. How troublesome this becomes will be determined by the computational resources available, and the frequency of updates.

Note that there may be multiple types of observations of a given system. For instance, an early application was, in fact, to navigation — specifically, ship-board systems in which an inertial navigation system subject to drift could receive periodic position fixes with much more accuracy based on landmarks or beacons of various sorts. The inertial navigation system allows the ship to maintain a position estimate by directly observing accelerations of the vessel, and by doubly integrating — but that estimate will, of course, be subject to drift over time due to noise in the accelerometers (e.g., gyroscope drift). The position fixes, by contrast, convey information on the ship's position itself directly — but are only available relatively rarely. The Kalman filter incorporates whatever information is available as it is available to produce a best on-line estimate of the system's state.

However, the Kalman filter does have certain limitations. To begin with, there is the difficulty that the system being modeled must be linear, or the Kalman filter equations do not directly apply to that system. Now, in the real world, nicely linear (or linearizable) systems are the exception, and not the rule. Therefore, many actual applications of the Kalman filter are to systems which are highly nonlinear. A common approach to modeling such systems is to use the so-called Extended Kalman Filter, or EKF, which is simply a Kalman filter based not directly on an accurate nonlinear model of the system itself, but rather on a less accurate linearization of such a model.

In effect, this requires the filter to treat the system's known nonlinearities as an unknown noise source for which it is required to compensate.

In fact, the state equations for many robot navigation problems cannot be linearized. In particular, this is always true if the state model explicitly models heading as a state variable (which is, of course, necessary if the model is also to deal with uncertainty in heading — a serious problem in many real-world environments), since the effects of the heading (a state variable) on position (another state variable or two) while the robot is in motion are highly nonlinear — they are, in fact, governed by trigonometric functions. The models of robot behavior which are linearized to form the EKF may nevertheless be quite elaborate — including terms for, e.g., change in wheel diameter (i.e., tire wear); however, they do, in practice manage to perform with pretty high accuracy.

2.6.2 The Kalman filter applied to grids

Having described the Kalman filter, let use return to the dual-grid robot mapping system of Schiele and Crowley [43].

Their robot is a wheeled car-like device with the sensors mounted on it. Now, a completely accurate state model for the robot is, of course, nonlinear, requiring the use of the EKF. The operation of the robot is, as described above, divided into discrete time intervals, each of which involves a transit, and some sonar readings during which a local grid is built and matched to the global grid.

There are two sources of position information in this system — odometry, which can convey updates and increase noise in position, and matches between the local and global grids, which may actually reduce noise. Each of these is incorporated into the robot's state using the Kalman filter. Finally, the global grid is updated. Note that simply summing the local grid data into the global grid would be an inaccurate reflection of what is known about the robot state, since the position of the robot is still uncertain. What is done instead is to first, in effect, convolve the local grid with the Gaussian filter represented by the covariance matrix of the uncertainty in position, and then sum *that* into the state vector.

In environments where there are a wide variety of obstacles at a wide variety of different orientations, this system works fairly well. However, there is a potential difficulty when the robot has to deal with environments which are more monotonous. For instance, suppose the robot is moving down a long corridor. In the direction across the corridor, it will receive continuous correction from its sensors. However, since one part of the corridor looks the same as another, in terms of the range data available (two walls at a particular set distance from each other), the robot has no way of knowing how far it has advanced along the corridor. Hence, the uncertainty in position in that direction will pile up, and features encountered at the far end of the corridor will be "smeared" in that direction. Depending on how long the corridor is, this may lower the density of detected object features considerably — perhaps enough to affect the ability of the system to match local a new grid to the smeared data.

What this reflects is a problem in the underlying representation. The grid represents the probability that there is an obstacle at a given location, or not — but it does not represent dependancies between locations. In particular, knowing that there *might* be an obstacle at each of fifteen different locations (as represented by fifteen nonzero entries in a Bayesian grid) is different from knowing that there *is* an obstacle at one of those locations, but we just aren't sure which. Yet the uncertainty we face is of the latter kind — the kind the grid cannot represent, because it supports no notion of obstacles as individual objects. It is therefore interesting to consider representations which do allow us to represent this sort of uncertainty, and reason about it, directly, ideally, using a sound probabilistic framework.

2.7 Beacon based systems

To summarize once again our problem, the robot is operating in an environment where it must deal with some measure of uncertainty both in its estimates of the changes in its own position from moment to moment, and such position readings as it may have on the obstacles in its immediate surroundings. Now, in the work previously cited, we have assumed that an individual obstacle cannot be identified as such — that is, all we can tell about a given obstacle is that it is at a given range and distance, and there is no extra information available about it which would allow us to determine whether or not this is the same obstacle we were looking at a little while ago.

However, some sensory systems are capable of identifying individual obstacles as such — the identifiable obstacles are often called "beacons" in the literature, even though they may (as we shall see) take a variety of forms. In practice, a wide variety of beacon types have been used, including sound emitters sensed by passive SONAR, reliable reflectors — such as corner reflectors — detected by active SONAR [28], bar codes on the walls which can be detected by lasers on the same principle as supermarket scanners, radar emitters which are detected by passive sensors on the robot [16] [17], vertical walls detected by processing range-sensor output [20] and so forth. Not all applications allow this sort of modification of the environment for the sake of the robot, but where it is permitted, it is certainly useful. Indeed, depending on the sensor arrangements, it may well be possible to find identifiable beacons which are reliably detected despite being passive features of the environment (e.g., corner reflectors for SONAR).

2.7.1 Simple applications of EKF: beacon-based systems

As it happens, many robot navigation systems based on detection and use of beacons are once again based on the Kalman filter. The application of the Kalman filter to this system is quite simple; the Kalman filter is used to estimate the state of of the robot as it moves in the environment based on odometric data. When a beacon is identified, it is used to provide a position fix, allowing the robot to locate itself directly based on the (now known) range and direction to the beacon.

The preceding paragraph assumed that if the robot detects a beacon, it is capable of identifying what beacon it is — which may be arranged, for instance, by making them distinctive bar patterns, or by having them emit a particular coded radio sequence. However, if this cannot be arranged, then the robot has to determine, once a beacon has been detected, which beacon it is. A common technique is to compare the position of the beacon that has been detected with the *expected* position of each known beacon relative to the robot, and regard the beacon as identified if it's position falls within a certain distance of one of the known positions.

The procedure of using a "validation gate" around estimated positions of beacons relative to the robot requires, of course, that the robot's position drift due to inertial or odometric error not drift so far that valid readings fall outside the gate. If that occurs, then the system becomes "lost" — the only available remaining sources of position data are drift-prone integration from inertial or odometric sources, which will only get worse with time. Thus, some of the robustness available in principle with the Kalman filter tends to be lost in these circumstances.

(It should be noted that the problem here is not with the use of the Kalman filter, or indeed probabilistic methods generally, *per se*, but rather with the limitations of one particular, well used probabilistic framework. It may well be possible to construct a probabilistic framework which can handle these difficulties; chapter 7 discusses some of the attributes such a representation *might* have, and whether the representations of space in robot brains may be seen as having some of those attributes. However, such frameworks are not yet common in the robot mapping literature).

The extension of the systems described above of most interest to us, of course, is when the robot is exploring an environment in which the positions of the beacons are not initially known. It may still be the case that the beacons can be uniquely identified. Indeed, this may well be the case in industrial environments — for instance, it may be a reasonable procedure in some environments for workers to slap barcodes on the walls which a laser sensor on the robot can detect, without themselves doing a detailed survey of their positions, and have the robot itself map out the positions of the beacons as part of the procedure of mapping out the environment. Such a sytems are described by several authors (see [4, 51, 35]). However, the general case is that the beacons are at unknown locations and unknown positions — for instance, Leonard, Durrant-Whyte and Cox [28] describe a system in which the beacons are reliable sonar reflectors which simply happen to arise in the environment (e.g., at corners).

Regardless, the procedure generally used for this case is to use a Kalman filter to estimate the position (in an assumed 2-d Cartesian frame) of each detected beacon, regarding each sighting of the beacon as providing further evidence of its position (by taking its displacement relative to the estimated position of the robot). Note that in these systems, the positions of the robot is estimated relative to the beacons, and the position of the beacons is estimated relative to the robot. There is a risk that this procedure will never converge; one can only say that in practice, such systems have been operated correctly at least under laboratory conditions.

(The procedure of using a "validity gate" to identify a new signal as originating from a known beacon is likewise somewhat more troublesome in this environment, particularly if the environment is such that two beacons may be close enough within the environment to be within each others' validity gate).

2.7.2 Vision-based beacons: Faugeras

A particularly interesting application of the Kalman filter to navigation in the general style outlined above is the system of Zhang and Faugeras [60], in which the role of the beacons in the systems described above is played by line segments detected in the visual environment by a stereoscopic vision system. The use of stereo allows the system to detect the range of the line segments directly. (Note that while some stereo systems attempt to fill in a range image of the entire scene within range of the cameras, this system simply takes the range of identified discrete features, which can be estimated with somewhat more confidence).

This system has managed to do quite good 3-d reconstructions of the layout of, e.g., a room with cardboard cutouts giving rise to line features painted densely on the walls. (Note that a blank wall would give rise to no features for this system, and hence to a hole in the map). However, the computational load posed by the system — running a Kalman filter (with all the matrix algebra) for *each detected line segment* in order to construct the validation gate for matching against subsequent segments — is considerable. (Furthermore, a somewhat complicated state is maintained for each line segment, which makes the individual filters quite a bit more complex than the simple model of a SONAR beacon indicated above).

2.7.3 Beacons without the Kalman filter

It is, of course, possible to use beacons without involving the Kalman filter, by invoking some alternative procedure. For instance, Chatila and Laumond [11], in their HILARE robot, use a procedure based on Bayesian reasoning to estimate the position of both the robot and the beacons it detects and incorporates into its map onto an assumed 2-d grid.

This system actually improves on the systems described above in at least one respect: It explicitly tracks the dependancies in the recorded positions of multiple obstacles. That is, if the robot encounters a beacon it has previously seen, it uses that information not only to correct its current estimate of its own position, but also the estimates of the positions of other beacons which it recently encountered. Many other systems treat simply the positions of beacons encountered as independant events, with no attempt to keep track of dependancies.

2.7.4 Problems with beacon-based approaches

Most beacon-based approaches are based on the Kalman filter, so it seems appropriate to begin by discussing drawbacks of the filter.

This computational load — several matrix multiplications and an inversion per sensor reading — is one cost of the use of the Kalman filter. If the only Kalman filter in the system is one modeling the robot's own position (with input from available sensor readings), it is not likely to be a severe problem, given the computational capabilities of even a fairly modest microcontroller. However, in systems which use a separate Kalman filter to model the position of each obstacle they track, and which simultaneously track a large number of obstacles, this cost can become considerable. An extreme case is the Zhang and Faugeras system, which runs a separate Kalman filter for each near-vertical line in view, and whose physical motion is ultimately rate-limited by the computation required for the updates.

Another point which might be raised is that the Kalman filter is itself based on certain assumptions which may not turn out to be true in practice. In particular, two key assumptions of the Kalman filter can easily go wrong — these are that the system itself is linear, and that the Gaussian noise model implicitly assumed by the filter (in the arithmetic it uses to combine covariance matrices) are both valid. However, as we have already mentioned, there are plenty of systems whose state equations cannot be rendered into the above form easily by any such trick, and unfortunately some of them come up very naturally in robot navigation problems. Indeed, a robot which can rotate, has some uncertainty in its rotational heading (and can also pick up such error as it travels forward) is exactly such a system. This nonlinearity inevitably extends into the secondary Kalman filters used by some systems to attempt to estimate the locations of the beacons.

The beacon-based models also make some assumptions about the properties of the beacons themselves, which may be troublesome in some applications. If we assume that the robot can distinguish each individual beacon from all the others reliably, we are obviously making substantial demands on the sensory capacity of the robot, which are difficult to sustain in the current state of the art without some alteration of the environment (placing identifiable beacons within it). However, if we do not, we have to avoid situations in which the robot confounds one beacon with another, which may also be difficult (for instance, if two beacons are, in fact, right next to each other in the environment).

2.8 Visibility-graph based systems

Regardless of how it is done, the business of estimating positions of identified obstacles on the ground plane (or rather, on the assumed ground plane — recall that realistic systems may have to go over hills) is complicated and awkward, enough so that one may realistically wonder whether there is a way to take advantage of the presence of single, identifiable obstacles in the environment in a manner which does not require exact estimates of their positions. Indeed, we might ideally like a representation which is robust in the face of at least small changes in the position of the obstacles. One technique which has been used toward these ends is the so-called visibility graph, which is defined in terms of certain obstacles which are designated, in terms of their suitability for use as landmarks.

The notion of a visibility graph is probably best illustrated at first in terms of a simple example. Figure 2-1 shows a simple environment with three obstacles identified as landmarks within it. Consider first the two landmarks A and B, and the line AB extending between and beyond them. The robot can clearly obtain at least a little information about its current location in the environment as a whole by determining which side of this line it lies on. This, in turn, is fairly simple to do — it can simply

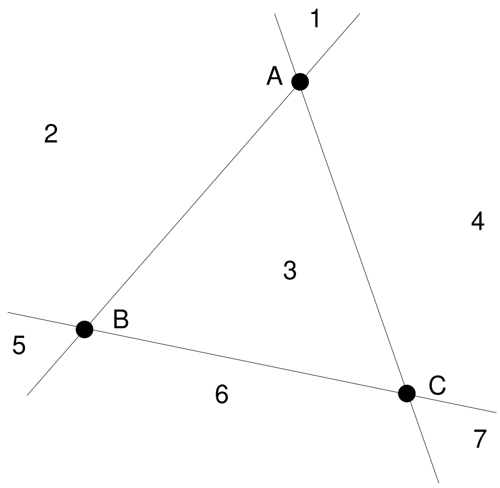


Figure 2-1: An environment with three obstacles. So long as all three obstacles are visible and are identified correctly, the robot can easily determine which side of the lines AB, BC, and AC it lies on. This in turn allows it to determine which of the numbered regions it occupies. We can then construct a graph with nodes for the regions, and links for possible transitions between them.

determine which of the two obstacles is to the left as it looks toward the mid point between them. This restricts possible positions for the robot to one of two half-planes — not a whole lot of information (one bit) but still information.

However, the same logic, of course, applies to the other lines in the figure, AC and BC. Thus, the robot can determine which side of each of these lines it lies on — which allows it to determine which of the seven regions delimited by the lines in the figure it occupies (one of which is even finite!). But, there's more. If we assume that the robot cannot actually walk right through a landmark, then it can only move between these regions by crossing one (and only one) of the lines. Therefore, we can construct a graph describing possible states of the robot (corresponding to the regions of figure 2-1), with the links between them illustrating possible transitions. The result is a graph which can be used as a map of this simple scene, known as the *visibility graph*. (These graphs are sometimes called aspect graphs — the terminology is borrowed from work in object recognition, in which the range of views from which a particular set of features of the target object can be seen is termed an aspect of the target object).

What is worth noting about this sort of a map is that it has some of the desirable properties which were elaborated earlier. If the robot's estimates of the positions of the landmarks are off by a slight amount, it will not affect the aspect graph at all, nor the sensing process which allows the robot to determine which cell of the aspect graph it occupies. Indeed, neither the sensing process nor the map itself actually uses metric information about the space which the landmarks occupy directly at all (though it does make other implicit assumptions about the space — for instance, that any given line extended out to infinity will in fact divide it into two disjoint rejoins, which is true for planes and spheres, but not for, say, tori). It takes major reconfiguration of the space (for instance, for landmark C to somehow cross the line AB) before the graph stops being useful for at least coarse navigation relative to this particular set of three landmarks.

Unfortunately, the representation is not so easily made robust against certain other likely types of errors, such as the failure to identify, or misidentification of, a landmark.

(Another thing worth noting in passing is that while having n pairs of obstacles in a setup like that described above might seem to allow for 2^n possible distinct place representations, given one bit of information — a factor of two — for each line, there will almost always be fewer. Indeed, we can see this even in the above simple example, as there are three lines, but only seven cells, not $2^3 = 8$. And indeed, enumerating all possible "sidedness" readings, we find that one of them is self-contradictory and can never occur).

In more complicated situations, the notion of the aspect graph gets more complicated, but the fundamental essence of it remains the same. We begin by defining some notion of a configuration of the visible obstacles, and an equivalence relation on those configurations. We then divide space up into sets of points with equivalent configurations ("cells") — in the example, these are points which are on the same sides of all the lines — and construct a graph which has nodes for these cells, and links for possible direct transits between them, for use as a map.

2.8.1 Basic applications

Kuipers' QUALNAV model [25] is a basic implementation of this idea, assuming individual, individually identifiable landmarks which can be picked up in the environment by some sort of sensor — radio or vision being the usual cases. (Much aspect-graph based research is done on outdoor environments, in which SONAR — many laboratories' sensor of choice, is difficult to use). Similar notions have also been pursued in simultation by Rao and Iyengar [39], and Simsarian et al. [45].

One interesting application of this system was by Talluri and Aggarwal [49], which operated in an outdoor environment, but used CAD models of the environment for the actual mapping (deriving the visibility graph from the CAD data offline — the model of the environment was presumed to be provided ahead of time. Features used were the tops of buildings as seen from the ground against the sky; these were assumed likely to remain stable).

2.8.2 Problems with visibility-graph representations

Visibility graphs are a representation which is strongly robust against small errors in estimating the positions of obstacles — precisely the sort of error which makes the "short-range" maps considered in the previous chapters difficult to maintain over long periods of acquisition. Indeed, the map stays unaffected even if the position of a landmark changes by a small amount, so long as its placement relative to other landmarks isn't affected "much" (to be precise, so long as it stays within the same cell of the aspect graph defined by all the other landmarks).

The major problem, as described above, is the difficulty with identifying the landmarks. This difficulty can be worked around in various ways — for instance, consider [52], whose landmarks, for a system to be discussed below which faces some of the same difficulties, are pieces of paper taped on the walls of his lab, with designs on them which are chosen specifically so that they can be easily segmented and classified. Likewise, if the robot is a planetary explorer, and the landmarks are radio transmitters dropped by the robot itself in the course of its operation and transmitting a unique signal, this difficulty would be no big deal.

It may be worth noting that in some industrial environments, this may be a reasonable way to build a complete system — many such environments *already* have bar unique bar codes on the walls at several points to support the systems which assure that the security guards are making their rounds, though those may not be dense enough for robotic purposes. However, for more "naturally occuring" landmarks — i.e., anything like those that might be used by humans, automated computer acquisition of the landmark set is very much an open research topic.

Another interesting problem with visibility-graph methods is the question of how to deal with the geometry of space *within* the space represented by a single node of the visibility graph. For instance, a single rectangular room with a door and nothing on the fllor would constitute a single cell in a strictly-defined visibility graph, since everything in the room (e.g., all features on the walls, doors, windows, etc.) would be visible from any location within it. We may, nevertheless, want the robot to have a notion of multiple locations within the room, so it could, for instance, plot a course from the door to a window.

Another problem is that the cells represented by visibility graph nodes are quite often of somewhat irregular shapes — e.g., long ribbons of space extending most of the way down the hallway, denoting a narrow region through which some very particular set of obstacles past the end of the corridor is visible. Routes defined in terms of such cells may have undesirable characteristics.

2.9 Location Hashing

As discussed above, the major practical problem with visibility-graph based systems such as those described above is the requirement that the robot's environment be filled with distinct, robustly recognizable landmarks at a sufficient density to support a useful map. This problem becomes even more severe when we require the robot to acquire the map "on-the-fly", by dint of its own exploration, though the severity of the problem will depend, of course, on the technology being used to represent and sense the obstacles.

Regrettably, attempts to use vision as the remote sensing mechanism responsible for detecting and identifying landmarks place us at the opposite extreme. While there are, at this point, systems which are capable of identifying at least some sets of landmarks in cluttered scenes fairly effectively (see [6, 7] for systems which work with a set of obstacles particularly chosen for certain robotic navigation applications, and and [42] for a wider survey), the automatic acquisition of a set of landmarks, particularly when those landmarks occur in cluttered scene and must be segmented out before being learned, is still very much an unsolved problem.

To avoid this difficulty, some systems try to build visibility-graph (or perhaps it would be better to say visibility-graph-like) representations by means of visual sense data which does not include relative positions of specifically identified landmarks. To understand how this might work, let us examine the notion of a visibility graph from a somewhat different perspective.

The visibilty-graph based systems described above work by examining the configuration of obstacles, determining something about the environment from the observations (specifically, on what side of a set of known lines the position of the robot happens to lie), and using the results to determine which of a set of defined regions of space the robot happens to occupy. They also keep track of the relative physical relationships between the regions so defined, by constructing a graph whose links indicate possible transitions.

The only part of this description which depends on the particular geometric formulation outlined above is the part in parentheses — that is, the use of *positions of the robot relative to certain lines* to delimit the regions of interest. If we can find another way to delimit regions which has the same geometric robustness, we may be able to arrive at a navigation method which has the same desirable properties, and perhaps does not place such extreme requirements on the sensing technology that they use.

In effect the positions of the robot on the side of the relevant lines are used as a code to determine the robot's location. So, what we want is another method of hashing the input visual data which will yield a reliable, robust identification, in the manner of a hash code. Ideally, this method should yield the same "hash code" for the same location regardless of lighting changes, small displacements of key identifying points, etc., while yielding reliably *different* identifications for places which are actually distinct (and not displaced from each other by some trivially small different). We will call these "location hash" methods.

A popular approach to constructing appropriate hash functions is by taking traces of sensor input of some variety — for instance, range profiles of the immediate environment gathered over a long traverse — and putting them through a clustering algorithm. Many of these algorithms involve neural nets — common approaches include the use of self-organizing Kohonen neural networks [26, 15, 50], and the system of Bachelder and Waxman, using ART-type classifiers in the tradition of work by Stephen Grossberg and his group [2, 3]. This, of course, requires the robot to spend a considerable amount of time traversing the environment (or at least an environment) in order to train the classifier, before it is possible to do any map learning. (The place representations can't be made usefully stable while the classification which they are based on is still in flux).

However, a complication with approaches such as this is that there is no guarantee that there will not be multiple locations in the robot's actual environment whose sensor readings will hash to the same value. For instance, two places could very easily just look alike, at least to the extent of having the same range profile. (Imagine several offices with a relatively standardized layout of furniture). Accordingly, the hash code cannot be used reliably, *by itself*, as an indication of where the robot has arrived in all environments.

There are various ways around this — for instance, Kurz [26] builds an explicit graph in which the nodes are associated not only with a sensory hash code (produced by means of a Kohonen net), but also estimates of the positions of the graph nodes. If there are multiple nodes at different positions with the same hash code — i.e., the same net-assigned classification code — the positions serve to distinguish them. (This obviously leads to ambiguities in the case of two neighboring nodes with the same hash code, but if that situation might arise, then Kurz's procedure will simply assign a single node; it only creates new nodes when there is a change in the hash codes. One wonders how the system would cope with, e.g., long corridors in which the range configuration of local obstacles is the same at every point; Kurz reports on no trials in such environments).

It is also interesting to note, in passing, that there are a few systems in the literature which have sensor systems specifically constructed to yield data which will yield good, reliable hashes. For instance, Yagi et al. [58] report on a system in which a camera is placed to look up onto a conical mirror, which reflects the surroundings from all directions into the camera in each frame; the hash function used is based on detection of vertical edges.

2.10 Graphs and topology

A common element between all of the schemes described in the preceding section was the construction of a graph as an abstract representation of conditions in the space occupied by the robot. All those schemes also focused on a particular method of constructing the graph, but there are alternatives.

In particular, for most of the systems described above, the graphs are tied to some parameterization (typically a 2-d Cartesian coordinate system) of the space occupied by the robot — though an exception might be drawn for the ART-based system of Bachelder and Waxman [2], whose nodes emerge solely from the operation of a neural classifier, and are not individually represented by any explicit data structure whatever.

However, it is certainly possible for a graph to be built based on other criteria — thereby representing the topology of the space without necessarily *directly* representing any of its metric aspects. That is, the graph may represent the fact that certain places are in some sort of a neighborhood relationship without necessarily representing the precise distance between them.

The obvious question, when building a system around such a notion, is what one uses instead of metric properties to organize the representation of space, determine what node is associated with the robot's current position, and so forth.

Two different approaches which have been tried in the literature are basing a system on the ability of the robot to recognize unique landmarks, and basing the graph transitions on actions which the robot has taken. Also, there is some interesting theoretical work involving learning automata which may be potentially relevant to actual systems.

2.10.1 Topological graphs in real systems

The first approach is taken by Taylor and Kriegman [52], whose system is based on recognition of stereotyped bar patterns on walls (though a more recent paper by Yeh and Kreigman [59] concerns approaches to finding similarly reliable features which naturally occur in, e.g., office environments). The system builds a graph in which each obstacle capable of being circumanavigated by the robot occupies a single node; neighborhood relationships are determined implicitly by constraints on the exploration strategy (which hence are obligatory as well on navigational actions which simply use the acquired map). This system records no metric information whatever about the relationships between the places so defined. (Drawbacks of the approach include, of course, the need for unique landmarks to be sufficiently dense within the environment).

The second approach was taken by Mataric [30], and somewhat later in a rather different form by Torrance [53]. Mataric's system was based on the subsumption architecture of Brooks [9], in which a robot's control system is built up of a group of simple autonomous processes which are capable of sending messages to each other, and maintaining local state. The messages are typically confined to be numerical values; this work was a reaction against the construction of large, explicit world-models and early subsumption-based systems were among the first to demonstrate that interesting behavior could in fact be sustained by systems which were entirely reactive.

However, the question of map acquisition is entirely a question of acquiring state (the state in question being the map itself). Mataric's system allows for this by having, in addition to the low-level processes supporting basic navigation and collision avoidance, a set of higher-level processes which function, in effect, as representations of locations in the environment. As the robot travels through territory which it has already mapped, these processes keep track of which one is currently active (and thus represents the robot's current location), and transfers to an appropriate neighbor when the robot moves into a differently characterized region. If the robot moves into a region covered by no existing process (i.e., graph node), a previously uncommitted one is recruited; this allows for new representation to be built.

2.10.2 Theoretical work

In addition to the work described above in actual robot systems, there has been considerable theoretical interest in the question of how a robot might learn its environment (and what formal framework one might set up to allow it to do so).

The most general approach to this question is to begin by noting that a robot in a finite environment with sensors of less than infinite precision may be usefully considered to have at most a finite number of possible positions and configurations (e.g., distinct headings) within that space — i.e., to have a finite number of states.

Thus, the system of the robot and the environment may be considered to be a finite-state automaton — and a deterministic one so long as the the outcome of any given action at a given state is deterministic (and has no random component due to motor error).

Accordingly, we can consider the problem of map learning to be reduced to the problem of learning a DFA — with variations according to whether the states are labeled or not (and whether the labels are all distinct), to account for the various possibilities concerning available navigational sensor input.

Indeed, there has been some theoretical work on the general problem of learning DFAs [40], which was indeed motivated by the goal of providing a theoretical framework for navigational learning. The attraction of this work is that it holds out the possibility of a map-learning algorithm which makes no assumptions about the environment whatever, whether metrical or of any other kind, and still functions effectively.

This work has, indeed, yielded algorithms which are successful in that, if they are allowed to monitor a given DFA for arbitrarily long, under certain conditions, they are highly likely to converge to a correct model of its state and transition graph. However, in cases where labeled states are sparse or ambiguous, the time required may be very long — e.g., high-order polynomials in the number of states of the automaton (which, for a realistic navigation application, may well be rather large).

Work in a similar spirit is reported by Dudek et al. [14] and Deng and Mirzaian [13]. Dudek introduced a paradigm in which the robot is considered as exploring a graph of unknown structure; this is similar to the DFA model. However, he extends it by allowing the robot to drop markers on the graph, thereby giving it a tool for resolving otherwise tricky ambiguities in the graph structure. Deng and Mirzaian extend this by considering the use of multiple markers. While this work is of a somewhat theoretical cast, the notion of dropping markers is not completely far-fetched — after all, animals do drop odor cues and thereafter navigate by means of them; Russel [41] describes a similar mechanical system which leaves odor trails which are mechanically detectable for some period of time, and which could presumably be use to drop discrete markers as well.

In any case, the behavior of actual robots in the real world inevitably does have a probabilistic aspect; rather than model them as DFA's (or determinstic graph-traversers), it might be more appropriate to model them as Markov processes, thereby allowing for the fact that the results of a given commanded action are *not* always certain. However, Markov process learning algorithms, while they exist, are just as bad from the combinatoric point of view as their DFA counterparts (as one might expect from noting that a DFA can be viewed as a Markov process in which transition probabilities are restricted to the values zero and one).

2.11 Review

As we have seen, there is a wide variety of approaches in use for long-term navigation. These systems vary in many ways; among the most interesting, however, are the differences in terms of what *kind* of information the various systems store about the environment. At one extreme, there are the modified grid methods which store nothing more than information about the apparent attributes of points at particular spatial coordinates. At the other extremes are systems such as that of Taylor and Kriegman, which do not actually store any metrical information of any kind — they build a graph which indicates topological relationships, but is not annotated with any information regarding the relative distances or directions between the obstacles noted on the graph.

Furthermore, these differences are, in at least, one respect, the most interesting differences between the systems in question. Two systems which use the same representation, even if they have a different learning strategy, will (if successful) learn the same sorts of information about the environment. However, systems which use different sorts of representations will, even if successful, learn *different things* about their environments, and that may have a material effect on how robustly those systems behave in varied environments (or, for that matter, in environments which change over time).

We have already seen a few instances of these sorts of tradeoffs. For instance, we noted in section 2.6.2 that a map representation based purely on the notion of Bayesian grids cannot directly represent — and hence, cannot reason directly about — the presence of an obstacle which is known to exist, but at an uncertain location. (It can only deal with the independant probabilities of an obstacle existing at each of several possible distinct locations).

Likewise we have noted, while discussing visibility graphs in section 2.8 that representations which are tied to the configurations of specific landmarks will acquire some measure of insensitivity towards errors in the placement of those landmarks. (Note that this advantage is somewhat degraded by location-hashing variants which identify places by their locations in some assumed coordinate system).

The contrasts which we have been discussing are, in effect, contrasts between different approaches to representing the geometry of the space occupied by the robot, and the geometry of the obstacles and landmarks within that space (though some of it also reflects the sensory capabilities, or lack thereof, of the robot itself — in particular, whether it has the ability to identify landmarks, or whether it can just detect the possibility of an obstacle in a particular direction). In particular, the methods described above differ widely as to how much information, and of what types, they require the robot to explicitly represent, ranging from a point-by-point representation of the space the robot occupies to a very abstract description of the topology itself and nothing else.

So, let us briefly consider different approaches to dealing with the space the robot occupies.

2.11.1 The bare minimum: getting topology right

Living with metrical information

To begin with, let us consider the use of metrical information — that is, of dead-reckoned estimates of the robot's distance of travel, and of direct measurements of the robot's position with respect to obstacles that it has detected. Metric information is, of course, crucial to grid representations of the robot's external space, but many of the other approaches described above try to keep it at some remove from the robot's actual operation.

If noise-free metrical information is available by some means, then its use renders the problem of reliable map acquisition trivial. And indeed, it may well be available in a properly prepared environment — for instance, differential GPS systems can give reliable position fixes to well within sub-meter range, which should be more than adequate for at least some of the robots used in outdoor applications (e.g., those built on modified automotive chassis). Likewise, if some oracle informs the robot in advance of the positions of some set of obstacles which it is able to identify reliably, then it will have reliable positional information (to within the limits of its sensors) so long as those obstacles are in view.

However, in many applications (e.g. indoor applications involving smaller robots), neither reliable direct position fixes and beacons at preset locations are not available. Hence the position of the robot must be inferred from information which is available — some combination of odometry and bearings to obstacles whose locations themselves must be inferred by the robot — all these observations being corrupted by the inevitable sensor noise.

Living without metrical information

In these circumstances, it is tempting to try to get as far away from direct use of metric information as we can.

Unfortunately, it is possible to construct realistic, and even fairly simple, environments in which a robot which takes no account of metrical information whatever will get the *topology* of the environment wrong, and hence be unable to effectively navigate. For instance, in 2-2, the distance traveled along the lower corridor is the only thing visible to the robot that distinguishes a case in which it has gone in a circle from a case in which it has not. (We will have more to say about this, and about similar cases in chapter 5).

Systems which build purely topological representations, for instance, that of Taylor and Kriegman, typically rely on the presence of identifiable beacons in the environment in order to be sure that they have the topology right, and in particular, to resolve precisely these sorts of ambiguities. However, this requires that beacons be present in the environment, and in such a manner that a given beacon can be identified on sight by its appearance. (Since we are trying to avoid the use of metrical information, the stratagem of identifying beacons by putting their apparent positions through a "validation gate" is obviously out of bounds). More than that: it requires the beacons to be sufficiently dense in the environment that at least one beacon is

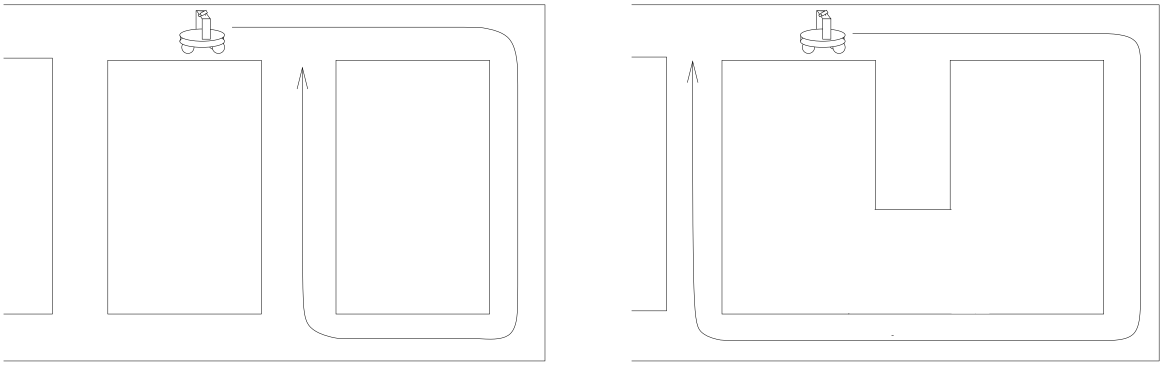


Figure 2-2: A situation in which odometric data has to be used to make a key decision. In one of these cases, the robot has gone in a circle; in the other, it has not. However, the only way to distinguish these two circumstances is by noting how far the robot has travelled along the lower corridor.

always visible to the robot from any location. (The robot has no way of telling "beacon-free" locations apart, so if it finds itself out of sight of a beacon twice, it has no way of knowing whether it has travelled in a circle or not).

2.11.2 Sometimes yes, sometimes no?

One final point deserves attention. While *global* geometry is difficult to recover in a noise-free manner, *local* geometry can typically be obtained fairly reliably, as the accumulation of error over short distances is likely to be small. Indeed, as we shall see, it can in fact, be quite useful in situations where no other source of positional information is available — as is often the case in at least one spatial dimension in corridors.

However, if this is one of our uses for local information, then paradoxically, attempts to assign consistent global positions to a set of obstacles may actually, in some circumstances, make the fit of local data to subsequent sensor readings worse.

For instance, suppose that the robot is in an environment which carries it over ramps. Even office environments, normally viewed as a simple case, may have these features. For instance, the third floor of the building housing the MIT AI lab, has ramps up to a false floor on one corner.

Any attempt to build a consistent set of obstacle placements based on the dead-reckoned position estimates the robot receives in this environment will run into trouble, even if the odometry noise is reduced to zero (and we have some way of knowing this). As a result, the robot will be unable to incorporate reliable and repeatable odometry data into its map not for any reason resulting from the data itself — not because of problems with the data, but because the data does not fit the assumption, built into the robot's world model, that the environment is a two-dimensional plane (because it simply is not).

It is worth noting, in passing, that the problems described above may be encountered for causes other than the presence of ramps in the environment. Surfaces which give rise to differing amounts of wheel slip (tile floors vs. carpet, for example) can give rise to very much the same effect. And of course, there is the simple accumulation of noise, which will always be present to some extent, unless the robot does, in fact, have a way of gaining a reliable position fix (through sightings of beacons at known positions, or through GPS-like means).

2.11.3 Conclusions and desiderata

Let us apply these considerations now, to our own robot. Its sensing consists of odometry and range information gathered by clever use of a camera; it has no capability to reliably *identify* beacons of any kind, save by passing, say, a detected corner in the range image through a "validity gate" to match it with a previously seen corner. This robot is used in office environments, but the actual data from chapter 3 show that even in those environments, it will sometimes be in situations where only one wall, or a corridor is in view.

Since office environments are typically full of corners which look pretty much alike, this is not adequate to resolve the "circle ambiguity" discussed in section 2.11.1. Some tracking of long-view metrical data will be needed for at least this purpose — but we wish to do so without compromising the use of local metrical data, and hence would prefer to do so without imposing a single global coordinate system on the robot's world view (which would necessitate alteration of valid data to fit the model in at least some real circumstances).

Indeed, one of the contributions of this thesis is an approach to the use of metrical data which achieves both these ends, in the form of a graph which includes local metric information without imposing the requirement that it be globally consistent with any imposed 2-d reference frame. The data structure as a whole is discussed in chapter 5. However, we begin by discussing the processing of the sensory range data into structures which we can use for the nodes of the graph, immediately following in chapter 4.

However, that long-range map is built on collection of data — in particular, of the range to nearby obstacles — using a camera in a somewhat unusual fashion as a range detector. The viability of that approach to sensing is best illuminated in a short-range context, so we begin in that context, and then proceed in subsequent chapters to describe our approach to building and managing the long-range map.

Chapter 3

Short-range maps: a simple vision-based approach

We now present a simple vision-based approach to forming short-range maps. The approach depends on certain features of the local environment, the same as required by Horswill's obstacle-avoidance system [21] — and in fact, it uses and elaborates on the sensing techniques from that system. For as long as the robot's odometry holds stable, it yields very high-quality results, though obviously this is not something which can be counted on for more than, say, five minutes or so at a time. (See section 3.2).

3.1 Basic ideas

We begin with the notion that the robot is operating in an office environment which satisfies the conditions required by Ian Horswill's thesis robot, Polly [21]. Specifically, Polly relied on the validity of the assumptions that

1. the floor is a plane
2. all obstacles lie directly on the floor
3. there is some property of the floor which is easy to detect visually (for typical office environments, it works to just note that it is one color and has no edges on it).

Polly also relies on the conclusion from these that the distance of an edge pixel from the bottom of the image is related by a monotonic function to its distance from the robot.

Given these assumptions, we can produce a rough-and-ready map of the robot's immediate surroundings, from a set of camera images of what is visible from the robot's position, as follows:

The camera on the robot is doing a rough perspective projection of the scene onto the imaging plane. So, let us define a robot-centered coordinate system on the floor, such that the y axis is immediately in front of the robot, and the x axis proceeds to its left and right. The z axis, in this coordinate system points straight up; note, however, that by assumption, all obstacles are at constant z; more precisely, they are, by assumption at $z = 0$. For generality, we assume that the camera is fixed at some unknown height Z above the floor, displaced by Y from the center of the robot, and inclined downwards at an angle θ.

Now, we wish to determine how obstacles on the $z = 0$ plane in our world coordinate system will be projected into the image plane. To determine this, we must compute the coordinate transformation from world coordinates system into camera-centered coordinates, and plug the result into the equations for the perspective projection. We begin by translating into a coordinate system with axes parallel to those of world coordinates, but centered on the camera:

$$\begin{aligned} x' &= x \\ y' &= y - Y \\ z' &= z - Z \end{aligned}$$

Note that we will always have $z' = -Z$ in practice, by the assumption that $z = 0$. We may then rotate these coordinates into a new coordinate system in which the y axis is parallel to the focal axis of the camera:

$$\begin{aligned} x'' &= x' = x \\ y'' &= y' \cos\theta + z' \sin\theta = (y + Y)\cos\theta + Z\sin\theta \\ z'' &= -y' \sin\theta + z' \cos\theta = -(y + Y)\sin\theta + Z\cos\theta \end{aligned}$$

Now, in this coordinate system, the standard equations for the perspective projection allows us to directly compute the image coordinates (u, v):

$$\begin{aligned} u &= \frac{x''}{y''} \\ v &= \frac{z''}{y''} \end{aligned}$$

However, since we cannot assume *a priori* that the camera geometry is known, it might be better to add the unknown displacements from the actual center of the focal plane to the origin of the (u, v) coordinate system, to yield the modified equations:

$$\begin{aligned} u + U &= \frac{x''}{y''} \\ v + V &= \frac{z''}{y''} \end{aligned}$$

Now, by a process of back-substitution, we can recover equations directly relating the world coordinates (x, y) to the image coordinates (u, v). As y depends only on v (and vice versa), we begin with them. Substituting, we have:

$$v + V = \frac{z''}{y''} = \frac{y\cos\theta + (Y\cos\theta + Z\sin\theta)}{-y\sin\theta + (Z\cos\theta - Y\sin\theta)}$$

From this, it is easy to derive that

$$y = \frac{Av + B}{v + C}$$

for *some* A, B, and C which are complicated combinations of the unknown parameters θ, Y, Z, and V. Similarly, for the x, which depends on both u and v, we have:

$$u + U = \frac{x}{-y \sin\theta + (Z\cos\theta - Y\sin\theta)}$$

whence it is, again, easy to derive that

$$x = (y - D)(\frac{u}{E})$$

for parameters D and E which, once again, are somewhat complicated functions of the original parameters of the geometry, θ, Y, Z, U, and V.

So, to summarize: we find that the real-world coordinates (x, y) of an obstacle are related to the coordinates (u, v) of the corresponding point in the image plane by a transformation of the form

$$\begin{aligned} y &= \frac{Av + B}{v + C} \\ x &= (y - D)(\frac{u}{E}) \end{aligned}$$

where the camera calibration parameters A, B, C, D, E are complicated functions of the height of the camera over the floor, Z, from the front of the robot, Y, of its angle from the vertical, θ, and of camera parameters u and v. Unfortunately, these measurements are difficult to perform accurately enough to be useful.

However, there is an alternative. Rearranging the above transformation, we see that

$$\begin{aligned} Av + B - Cy &= yv \\ Ex + Du &= yu \end{aligned}$$

These equations are nonlinear in the various coordinates, $(x, y), (u, v)$, but they are linear in the calibration parameters. Therefore, if we have a large set of known points (x_i, y_i), and the corresponding image coordinates (u_i, v_i), we can solve for A, B and C by doing a simple least-squares regression on the system of equations

$$Av_i + B - Cy_i = y_i v_i$$

and likewise solve for D and E by

$$Ex_i + Du_i = y_i u_i$$

As a practical matter, the easiest way to acquire an image containing a large number of obstacles at known offsets from the robot was found to be placing the

Figure 3-1: Camera calibration images — these contain numerous detected obstacle points at known offsets, placed with reference to the regular grid of an ordinary tile floor. Left image from the robot Wilt; right from Elvis.

robot in a room with a tiled floor, and putting black legos at intersections of the tiles. So long as the robot is itself positioned carefully with respect to the tile grid (in particular, some care is required to be sure that the camera's focal axis is parallel to the tile intersections), it is easy to acquire images through the robot's camera such as that in figure 3-1, which have obstacles at positions which are all immediately known.

To summarize: given our assumptions about the environment (most importantly, that obstacles lie on, or near, the floor), we have developed a transform which allows us to determine the real-world displacements of visible obstacles from the robot itself. By taking advantage of odometry, we can integrate observations over brief periods of time, producing a map of all obstacles visible in the local environment as follows: The robot's odometry gives, at any time, an estimate of the robot's current position and heading, *i.e.*, the position and direction of the camera. These may be combined with displacement to the obstacles to give an estimate for position of each obstacle. Then, we simply quantize these coordinates, and incrementing the corresponding cell of a map array, in the manner of a Bayesian certainty grid, such as are commonly used in sonar mapping robots. (However, the grid here is much smaller; adjacent grid points are roughly one inch apart).

3.2 Results

So much for theory. The approach to short-range map-making discussed above has been tried out in two robots, Wilt and Elvis. Both robots are based on commercial, omnidirectional RWI robot bases; aside from the dead-reckoning capabilities of the robot base itself, their sole sensor is a camera, which is mounted roughly above the center of the (circular) robot base, and tilts downwards to get a better view of obstacles on the floor. (The robot base also registers the current being delivered to the motors; a spike in this value can be used as a crude collision detector. However, the obstacle avoidance routines work well enough that collisions don't often occur in practice, and the mapping system does not attempt to make any use of this information; indeed, the Elvis robot code does not make use of it even for collision detection).

Wilt's camera is mounted on a tall pole, and is about four feet above the floor;

its imaging element is a fairly crude 64×32 CCD, mounted behind a lens. Elvis's camera is mounted on a bracket, and is only a foot and a half above the floor; it is, however, a much better camera; it actually delivers video at full NTSC resolution, but is subsampled down to 128×64 (and converted from color to monochrome) before running the edge detector, as higher resolution is simply not useful in practice for this application (known sources of error leading to misestimations of position considerably larger than a single pixels' diameter, as discussed below).

The simplest test of the short-range mapping approach is to place the robot in a large, enclosed space, let it follow its usual obstacle-avoidance routine, and build a short-range map by superimposing the computed obstacle positions for several successive frames, offset by the amounts indicated by the robot base's own dead-reckoned position estimates (and uncorrected by any visual data). The data is superimposed by initializing the map to be a plane which is entirely black, and incrementing the gray-level of each pixel corresponding to real-world coordinates at which an obstacle is detected on each frame where an obstacle is detected there. In practice, it is useful to subsample the frames, as the obstacle values quickly saturate otherwise.

Note that for these simple tests, the robot does not have to build the map internally; it suffices to read the raw obstacle-height data out of the robot over a radio-modem, and process it offline.

Initial experiments were conducted using Wilt. An image of the maps achieved in this test is shown in figure 3-2. Note that the general configuration of the furniture that the robot had to maneuver between is easily visible. Indeed, the result looks, quite literally like a map of the environment; while there are errors, discussed in detail below, the pieces of furniture and the walls which define the robot's reachable space are all visible, and all in their correct spatial relationships.

A better view of the limitations of the technique is seen in the image of figure 3-3. This is from the robot Elvis, traversing the same space, over a more extended period of time. There are two obvious differences between this map and the last. The first is that the view of the walls, etc., is less complete. This is simply because the camera on this particular robot, being close to the ground, has a more limited effective horizon.

What is more interesting is the emergence of mapping errors due to errors in the underlying dead reckoning. This can be seen in locations which were revisited multiple times by the robot, which appear several times in the map, slightly offset. This is a direct, visual representation of the accumulated dead reckoning error, which can be seen to be in the range of about half a foot over the three minutes or so which it took the robot to make a complete circuit of the playroom. Note that the slip seen here is for the robot operating on a carpeted floor, which is a surface known to be particularly conducive to wheel slip and navigational noise.

3.2.1 Characteristics of the short-range maps

As we have seen above, the major source of error in a map produced by this subsystem *appears* to be the accumulated error in the robot's dead-reckoned position estimates. An analysis of the other observable contributions to errors in mapping helps to demonstrate that this is, in fact, the case.

Figure 3-2: Map of the playroom, as seen by the robot Wilt. Small isolated dots indicate robot position fixes, and hence, the general path of the robot. A map of the space being traversed is at right.

Figure 3-3: Map of the playroom, as seen by the robot Elvis. A map of the space being traversed is at right.

3.2.2 Errors in the mapping process

Generally, it does seem to be the case that when the robot is put in the same place, and the lighting conditions are even approximately the same, the same distance readings will come out. There are two reasons why the resulting readings might not correspond to reality: the robot could fail to recognize a true obstacle (or hallucinate a false one), or an obstacle which was correctly perceived could have an incorrect range computed for it. We will call these misperceptions and incorrect ranges.

Misperceptions

The typical ways that false obstacles occur are as laid out in Horswill [21]; these include spots of bright light, from reflections in tile floors or sun shining in from doorways, which may be read as spurious obstacles. An optimal environment is therefore one with controlled, industrial lighting, and floors which are not themselves specular reflectors (and which, of course, have no designs on them which might trigger the edge detector); in such an environment, hallucinations are typically not a problem.

Failure to image an obstacle is much rarer — the obstacle in question must somehow manage to not trigger the edge detector. This has been observed in certain relatively rare situations — most typically a table or chair will have a reflective leg, which, from certain angles at least, reflects the floor it is standing on exactly enough that the edge detector simply does not detect it. In these situations, the match is generally exact enough that the human eye does not detect the obstacle either, except by context (the cues to the obstacle's presence being about as subtle as the cues to the existence of a glass door — which, indeed, humans will sometimes fail to perceive).

Incorrect ranges

Given that an obstacle has been imaged correctly, it might still be misplaced on the short-range map. This can, of course, only occur if the assumptions underlying the math above are somehow incorrect. The major ways this can happen are as follows:

Camera Distortion The lens of the camera does not in fact implement a precise perspective projection. Fortunately, this has not proved to be a problem in practice, because most obstacle points seen are in the central region of the camera where divergences from true perspective tend to be small. Camera distortion tends to be least, within a few pixels, near the center of the image, and to increase towards the margins and especially towards the corners. Since few obstacles are encountered on the extreme right or left of the image plane, and none towards the top (above the horizon, there can be no obstacles), the obstacle pixels affected most by camera distortion will tend to be those on the bottom of the image plane.

Of course, the significance of a few-pixel error will vary depending on where the displaced pixel falls. As shown below, given the transformation derived above for transforming image coordinates to real-world obstacle coordinates, the magnitude of the error in position induced by a one-pixel displacement on the image plane is the least for obstacles which are close, and which are imaged towards the bottom of the image plane, where camera distortion becomes most significant.

Inaccurate modeling of the robot geometry The analysis above makes an implicit assumption that the camera bears at a fixed angle with respect to the environment. In fact, the camera geometry may change over time — for instance, if the floor isn't entirely flat, the robot could be on a tilted section, which will mean that its angle relative to true horizontal is not the angle which was calibrated. (It is worth noting that it is often the case that floors vary somewhat from the horizontal, due to slight variations in concrete pours during construction, and that this can be quite difficult for humans to detect).

The most extreme degree of tilt to be expected in normal circumstances is on the order of five degrees.

It is also worth noting that the camera mounts on all robots used for this work tend to vibrate somewhat as the robot moves; this may lead to additional minor changes in camera angle, and may also contribute to the phenomenon of "blur" in the maps discussed below.

Inaccurate modeling of obstacle gemoetry

The position of an obstacle may also be estimated incorrectly because our assumptions about the obstacle itself may not hold. Put simply, the thing may not be sitting on the floor. The most obvious way for this situation to arise is for the obstacle to be something like a couch which is, in fact, raised somewhat above the ground except where its legs make direct contact. However, there are other situations in which it also occurs — for example, in the sample office environment in which most of our actual tests were conducted, the edge detected for a wall was typically *not* the intersection of the wall and the floor, but rather, the top of a modling, a few inches off the floor, which had been all too carefully color-coordinated with the carpet.

Errors in the edge detector

The placement of an obstacle in this scheme is, of course, computed from the placement of the edge pixel that that obstacle produces. However, interaction between the edge detector thresholds and pure camera noise causes the positions of corresponding edge pixels to vary from image frame to image frame.

By and large, errors due to this cause will amount to only a pixel or two, generally corresponding to an error in map coordinates of a few inches at most. However, these errors will tend to vary somewhat from frame to frame, due to variations in the imaging process (i.e., camera noise), with consequences which are discussed below.

Magnitude of these errors

The major causes of these sorts of error, as discussed below, are typically more associated with errors in the imaging process, or our modeling of it, than with errors in the position per se. It is therefore interesting to determine how much linear error can be caused by the misplacement of the image of an obstacle by a given number of pixels.

The actual values for the parameters, in the transforms above are:

$$\begin{aligned} A &= -22.87 \\ B &= -1177 \\ C &= -49.69 \\ D &= 31.86 \\ E &= 118.1 \end{aligned}$$

for the robot Elvis. That is, we have roughly:

$$\begin{aligned} y &= \frac{23v + 1177}{v - 50} \\ x &= (y + 32)(\frac{u}{118}) \end{aligned}$$

for image coordinates (u, v) in pixels (in a 64×128 image), and (x, y) are obstacle coordinates, in inches.

The horizon in these images is located at roughly $v = 50$, explaining the singularity in the coordinates. This transform gets arbitrarily sensitive near the horizon, as this singularity is approached. However, closer to the robot, the consequences of a one-pixel error diminish quickly — eight feet from the robot, a one-pixel error is six inches; at five feet, three inches; at four feet, 1.5 in. Most obstacle readings are from the closer distances, typically within four feet.

The magnitude of an error in computed x position for a given horizontal image coordinate u also depends on v (though not on u itself). At eight feet, a one pixel error gives rise to a final error value of about an inch; at five feet, rougly 0.78 in., and at four feet, 0.68 in.

Significance of these errors

Regardless of magnitude, one thing which all these sources of error (except for noise in the edge detector, which is, again, discussed below) significantly have in common is that they are repeatable. Therefore, if the robot is in the same location, none of these error sources will keep the robot from perceiving obstacles at the same relative displacements. As we shall see, the major use of these short-range maps in the complete system is to enable recognition of locations by the configurations of obstacles which are visible, so that too will tend to reduce their impact on the functioning of the system as a whole.

An even more significant fact about all of these errors is that they tend to be small, particularly when compared to the errors induced by positional slip over the course of several minutes of continuous operation.

3.2.3 Other characteristics of results

One final thing which is obvious simply on looking at example short-range maps of this form, but which is worth discussing explicitly, is this: they are blurry.

The major cause of this blurring is that, as discussed above, the camera noise will, cause corresponding edge pixels (delimiting the boundaries of the same obstacles) to be displaced very slightly from frame to frame. This results in the short-range maps, which are accumulations of several frames' worth of data, accumulating, for any given detected obstacle point, a "cloud" of observations centered on the most commonly detected location.

This blurring can be seen as a noise phenomenon — however, noise can be useful, and indeed, we do exploit it quite directly in the long-range mapping system, as discussed in chapter 4.

3.3 Conclusions

To summarize — we have derived a system by means of which a visually guided robot in a common office environment may acquire mapping data concerning the obstacles immediately surrounding it. The errors which the system makes in mapping a small local area are themselves generally minor, and not likely to significantly interfere with the functioning of the system.

However, this system does not, by itself, correct for the accumulated slip in the robot's position estimates, which it relies upon implicitly, and which do have errors of their own. The existence of these errors is one of the main reasons why the short-range mapping system cannot, by itself, be the solution to the mapping problem, and is the major problem which our long-range system attempts to deal with.

However, the magnitude of these errors sets certain limits on the problems which the long-range system needs to solve. In particular, suppose the robot explores a particular place, and builds up a representation of it. If it then explores the surrounding area for several minutes, and then revisits this place which it had already mapped, the accumulated positional slip is likely to cause a degree of error which would displace with respect to that representation, but not by more than the diameter of the representation itself. Our long-range mapping strategy will take advantage of this

To put it another way, if the robot makes an extended traverse, and the dead-reckoned displacements around that traverse sum to zero, it cannot then conclude that it has gone in a circle; in fact, it is almost certainly displaced by some distance from its initial location, and some sort of position fix based on external reference information will be needed to sort out exactly where it has arrived, and how it relates to previously visited locations.

On the other hand, over *short* distances, the robot's dead reckoning may well be at least as reliable as any other means of estimating its location — even if there are features that can be picked up by the above short-range mapper (or, for that matter, other means). For instance, suppose the robot is in a long corridor, where the sidewalls of the corridor are in visual range, but the end of the corridor is not. In this situation, dead reckoning is the only way the robot has of determining exactly how far along the corridor it has advanced.

To summarize, broadly speaking — it can be useful (and may even be necessary in some cases) to rely on dead reckoning to determine the distance between two locales. However, as the robot moves, error piles up. Hence, as the distance that the robot has traversed between two places grows longer, the dead-reckoned estimate of the distance between those two places becomes less reliable. Note that the distance of the traverse may be very different from the estimated *displacement* between the endpoints of the path — for instance, if the robot's internal odometry shows it has moved in a circle, the estimated displacement of the endpoints of the path is (by definition) zero, while the distance of the traverse (and hence, the error in the dead reckoned estimate) can vary from small (for a quick out and back) to arbitrarily large (in the case of an extended period of operation).

As we shall see, this tradeoff conditions our use of odometric information in the construction of more stable maps. We do use odometric data, but we limit its use to keeping track of metric relationship between a place and its immediate neighbors. We do not use dead-reckoned estimates of displacements gathered over long traverses at all unless

- They indicate a short physical displacement, and ...
- they do so in a matter which can be subject to a direct visual check.

In the chapters 4 and 5 we discuss a strategy for dealing with this class of problem.

Chapter 4

Long-range mapping 1: Place Representations

Let us now consider the problem of building long-range maps in the context of our robot. Recall that the goal of the work is to build a mapping system which works, and works stably, given a robot with a camera, semi-reliable odometric feedback from the robot base, and as little else as possible.

In chapter 3, we saw that it is possible to provide fairly stable short-range maps which are very good for as long as the robot's odometry remains stable, which is on the order of a few minutes (and a few hundred meters of motion) even under poor conditions. We would like for our long-range mapping strategy, whatever it is, to require no more. Indeed, we would like it to use the same raw data, if possible, both for reasons of parsimony, and because we already know that those features can be reliably extracted and used in our target environment.

Now, what the short-range mapper produces is, in effect, a fairly dense range image of the scene, giving the range to the nearest obstacle (if one is visible) in all directions, so that is the data we have to work with. We now examine two approaches to attempting to use it, one of which did not work out, the other of which did.

4.1 Place representations — an outline of our strategy

Fundamentally, the long-range map has to be a data structure which needs to support four operations:

- Storing information about a location the robot occupies.
- Matching the robot's current position to other locations it *might* be occupying, and estimating the accumulated drift in the robot's dead-reckoned position estimate if it has indeed returned to a previously seen place.
- Suggesting useful directions for exploration.
- Allowing the robot to plan paths from its current, known position to some position which it has occupied in the past.

Furthermore, it is best for our purposes if the map represents local geometric relationships without tying them to a single global coordinate system. The reasons for this have already been briefly discussed in our review of long-range mapping systems. (To summarize, if the robot returns to a previously seen place from a novel direction, and has accumulated substantial position drift in the meantime, it can be difficult to come up with a strategy for distributing the observed error along the loop which the robot traversed to get back to its original location. Also, in cases where the environment introduces systematic errors, a single consistent global reference plane may not exist. However, the local relationships are consistent and reproducible; they are in any case the only things that we can directly observe anyway).

A data structure which meets these requirements is a graph of discrete representations of individual places (and perhaps of obstacles as well), where the links represent known neighborhood relationships. Our system uses just such a graph. The strategies used to update and maintain it are detailed in chapter 5. Our concern, for the moment, is with the nodes.

The information required at the nodes, then, is how to represent the obstacles surrounding the robot at a given location, in such a way that the robot can tell whether it has returned to the vicinity of that location, and if so, estimate its displacement.

We tried two approaches to this problem. The first was to build detectors for commonly occuring features in the environment — particularly long lines and corners. The notion was that these would be ultimately embedded in a graph along with place representations, though the initial work just tied them to a global grid, and the idea was rejected before it could be elaborated into a full graph structure. Nevertheless, the reasons for the failure are interesting and instructive, and accordingly, we discuss it at some length.

A more successful approach, which was ultimately used in the final system, was to construct representations which were much closer to the raw data, and simply represented the configuration of the free space around the robot in a particular place. We begin (as we did in the actual experiments) with the feature-detector approach.

4.2 Detection of predetermined features

The initial approach considered for place representations (or indeed for any sort of correction of raw odometry data) was the feature detection approach. This was very much in the spirit of the feature-detection and kalman-correction based approaches of, e.g., Leonard et al. [28] and Laumond and Chatila [11], with features based on the visual data, rather than the sonar sensors used by those systems. In the end, it proved difficult to provide a set of features which could be recognized stably enough, given the abailable data, to work effectively. Still, the effort is worth chronicling.

In this work, the robot was programmed to detect features of a sort which occur fairly frequently in office environments, specifically, long linear constellations of obstacle points (corresponding to walls), and corners between recognized walls. (The appropriateness of wall features was suggested by their use by Mataric in her work on the robot Toto [30]).

4.2.1 Lines

Line recognition proceeded in two stages; first, line primitives are recognized in individual frames. The robot then used this steady stream of primitives to map out where the physically stable linear objects in the environment actually are — basically, by finding and collections of primitive line features which are continuous in space, and stable over at least short periods of time.

Extraction of line primitives

Recall that the short-range mapping approach which we outlined in chapter 3 was based on an "inverse perspective" projection, which maps obstacle points in the image into an image of the robot's "region of safety", that is, to the boundaries of the region in which the robot can move without hitting anything. If we impose an order on those points, say right-to-left, we then have a sampling of the curve which bounds that safe region, and we can compute a polygonal approximation to that boundary curve. Long linear obstacles should generate long sides in this polygon, and these are what we use as line primitives.

There are two complications to consider in this process. First, the boundary of the safe region may not be entirely in view — some of it may be "over the horizon" and out of the robot's view. So, for each frame, the robot actually considers several curves of obstacle points. Each such curve has regions which are out of view on either side; a point may be out of view either because the nearest obstacle in that direction is over the horizon, or simply because the camera can't see in that direction.

Second, since perspective projection maps lines to lines, we can actually run the polygonal approximation routine on the image coordinates of the obstacle points, as opposed to their reconstructed world coordinates. In theory, this should make no difference whatever. In practice, it does — reconstructed world coordinates are sensitive to errors in the camera calibration, and image coordinates are not. (Besides, working with the image coordinates yields a faster, simpler piece of code).

Line features

The raw data on which a line feature was based, ultimately, is the world coordinates of the points that went into it. The slope and intercept of the line itself are computed from these coordinates by a weighted linear regression, with obstacle points farther away from the robot given less weight (because the camera calibration is assumed to be worse for distant objects).

One of the advantages of linear regression is that it can be done incrementally — once we have the first- and second-order sums of the coordinates, we can discard the coordinates themselves, and still be able to revise the estimated position of the line when a new set of points comes in (by adding their coordinates into the old sums, and recomputing a new slope and intercept in terms of the result).

Besides these sums, the system does keep additional information on each line, such as the location of the endpoints. Also, when adding points to a line feature, the system adds the number of times that an obstacle was seen at those world coordinates (from the bitmap map) to a running count for the line feature. By summing the amount of support for the individual points, the system gets some idea of the amount of support for the line; this, in turn, is used to determine whether or not to keep the feature after its grace period has run out.

Finding stable obstacles

No matter how they are found, the primitives which come out of the image-frame line detector are basically sets of obstacle points. The robot is now faced with the task of figuring out which of these represent stable obstacles, and which simply represent confusion on the part of the feature detector. Those primitives which do correspond to stable obstacles must, in turn, be combined to form some notion of where the stable obstacles actually are.

To restate the problem faced by the robot — it has a set of remembered obstacle line segments, and a new set of primitives; it wants to compute a revised set of obstacles which hopefully corresponds a little better to reality.

It begins by constructing a new line feature for each primitive. It then searches among the obstacles which it already knows about for good matches to those primitives. If it finds such matches, it combines them with the new primitive to form an updated estimate of the obstacle's position. If there are no matches to a primitive from the set of known obstacles, it is added to the set on a tentative basis — if there isn't enough support for the new obstacle after a grace period has expired, it will be dropped.

Matching primitives to previously seen features

The system used three criteria to determine if a new line primitive was a good match to a preexisting line. These criteria are best expressed in a coordinate system in which the x' axis is placed to coincide with the feature that would result from merging the two lines. In that coordinate system:

- The center points of both lines must have y coordinates whose magnitudes fall below a threshold (i.e., the lines can't be displaced by more than a set distance along their perpendiculars).
- The size of the x range covered by the two lines must exceed a set threshold (there should be some overlap along the range covered by their parallels).

Finally, the orientations of the lines have to be sufficiently similar; this is checked by taking the dot product of two normals (which are stored precomputed with the line features) to get the cosine of that angle, and thresholding the cosine.

4.2.2 Corners

In addition to the lines, we built a detector for corner features. Corners were thought to be of some use in estimating the robot's position, as a distance and direction to a single corner uniquely defines the robot's position with respect to the map, so long as the robot knows which corner it is looking at.

Corner detection worked off the set of lines which the robot had built. We had the robot detect lines with endpoints which were close enough to each other, and which were close enough to perpendicular. The pair of endpoints then essentially defined a corner feature. The only parameters of this corner detector were the maximum allowed distance between the endpoints of two lines forming a corner, and the lower bound on the angle between their orientations. (Again, this was actually checked by computing the dot product between normals to the two lines to produce the cosine of the angle between them, and then thresholding the cosine).

4.3 Fine Correction of Odometry

As outlined above, the ultimate point of creating these feature representations was to be using them to check the robot's current position with respect to the map, and correct its internal estimates.

We begin by matching image features to world features as described above. We then correct first the robot's heading, and then its position, based on those pairings.

4.3.1 Heading Correction

If a pairing between an image feature and a world feature is correct, then the difference in location between the two should, in principle, be zero. If not, then the sine of the angle between them can be computed by taking the dot product of a unit normal to one line with a unit parallel to the other. (Since the normals are stored precomputed, and the coordinate transform to turn them into parallels is trivial, this is quite fast).

Computing the arcsine then gives us the angle between the two lines. However, since this angle is presumably small, we have $\sin x \approx x$ (and using this approximation not only saves us the trouble of actually computing an inverse trig function, but has the salutary effect of deemphasizing outliers).

The average corrections for all pairs (a numeric average, since the angles are all near zero) was multiplied by a gain and applied to the robot. (As indicated below (section 4.3.3, a more sophisticated approach involving Kalman-filter like techniques for estimating the reliability of individual features might have proven more successful).

4.3.2 Position Correction

Computing the position correction was more complicated than computing the heading, because we want to correct two degrees of freedom (robot x and y), but each pair of lines can really only give us information about one degree of freedom (roughly speaking, along the normal).

So, we are effectively trying to compute a displacement $\mathbf{d}$ which minimizes something like

$$\sum |(\mathbf{d} \cdot \mathbf{n}_i) - \mathbf{d}_i|^2$$

where $\mathbf{n}_i$ is the normal to the i'th world feature, and $\mathbf{d}_i$ is the displacement between the center of the corresponding image feature and the closest point on that line. If everything were exact, the minimizing $\mathbf{d}$ would specify a displacement which precisely moved the image features onto the world model.

In fact, given the sums which we are keeping in order to do the incremental least-squares updates for feature merges, we can do better — the corresponding sum over all the input points which went into forming the image line is:

$$\sum \sum |(\mathbf{d} \cdot \mathbf{n}_i) - \mathbf{d}_{ij}|^2$$

where $\mathbf{d}_{ij}$ is the distance from the j'th point to the i'th line.

Now, the $\mathbf{d}_{ij}$s are impossible to compute directly from the available data, since the coordinates of individual points are forgotten once line features are formed. However, we do still have the first and second moments of their coordinates. That is, if each point $\mathbf{p}_j$ has coordinates (x_j, y_j), then we have $\sum x_j$ and $\sum y_j$ (as well as some second-order moments which we don't need for this particular purpose); these sums are sufficient to compute $\sum \mathbf{d}_{ij}$ which is all we really need.

Specifically, let $\mathbf{o}_i$ be a point on line i; it's convenient to use $(0, b_i)$ where b_i is the line's y intercept. Then, we have

$$\mathbf{d}_{ij} = \mathbf{n}_i \cdot (\mathbf{p}_j - \mathbf{o}_i)$$

and

$$\sum \sum |(\mathbf{d} \cdot \mathbf{n}_i) - \mathbf{d}_{ij}|^2$$

can then be expressed in terms of the moments.

4.3.3 Feature Extraction — results and problems

The above-outlined feature detectors were tested on data gathered from the robot Wilt, and processed offline in a Sun SparcStation, prior to the acquisition of the robot Elvis for the project. Results achieved for an extended trace, involving a trip most of the way across the playroom, and back, are shown in figure 4-1.

Unfortunately, the feature detectors proved to be unreliable in practice, with the result that the approach was abandoned in favor of the simpler and more robust scheme oulined in section 4.4. We now briefly discuss some of the problems encountered.

Inadequate world model

The most fundamental problem encountered was that there are more things in the world than straight lines and corners, and confronted with some of those things, the feature detectors tended to misbehave. For example, the sample environment contained a large circular trash can. As shown in figure 4-2, the response of the line feature detector was to divide the trashcan up into small linear segments. The problem with doing this is that the loci of the boundaries between the segments obviously depends on the angle with which the robot approaches the obstacle. So,

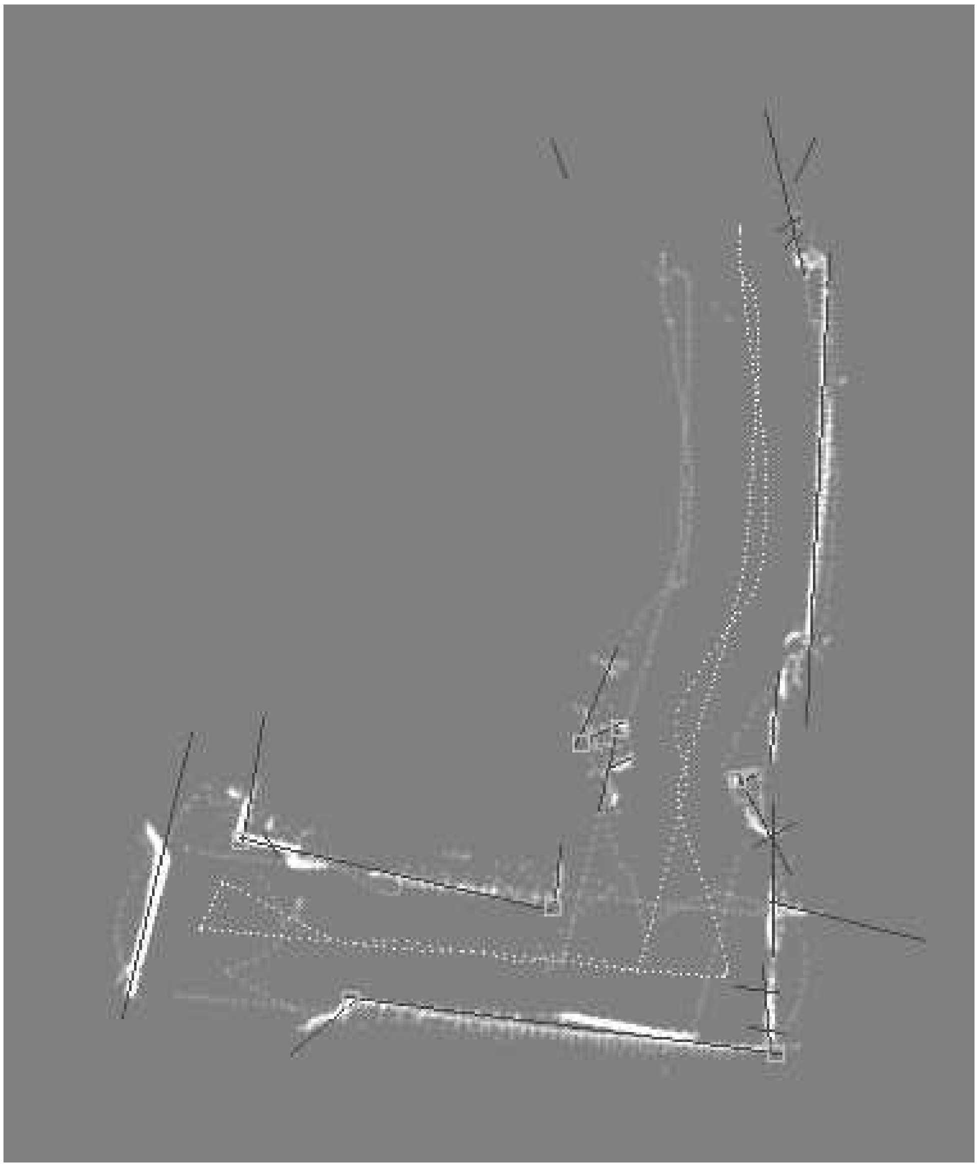


Figure 4-1: Results of the feature detectors on data from the robot Wilt. Data were collected on a trip across the playroom, to the opposite corner, and back.

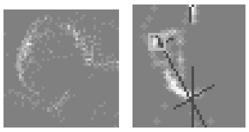


Figure 4-2: Feature detector misbehavior — a large round obstacle (in this instance, a trash can) is split up into segments in a fashion which is not robust.

when the robot returns to the vicinity of a round obstacle, it is likely to segment it a different way then when the obstacle was first encountered, which will pose problems when the robot tries to reconcile two incompatible descriptions.

Similar difficulties were encountered with such obstacles as office chairs with rolling bases. Typically, these chairs have bases with several arms, and with a caster at the end of each. These features are, physically, straight lines, so the difficulties encountered with the circular obstacles don't trouble the robot with the chairs. However, depending on the angle of approach, many of the arms of the base wound up being extremely foreshortened (which ones depending on the angle of approach). As a consequence, these arms were frequently missed by the line detector; furthermore, when they were not, there was generally not enough data to establish their organization, which lead to numerous undesirable mergings with the features corresponding to other base arms. So, while the cause of the problem with these obstacles was different, the effect was the same: the obstacles resulted in a set of features which was not stable as the robot moved, and so could not be reliably matched with prior observations when the robot returned to a different location.

Difficult to set parameters

Another difficulty encountered with the line feature detector in particular was that it proved difficult to set the thresholds for it in a manner which resulted in the desired behavior in all cases. For instance, in the seventh floor playroom environment during the test run, there were a couch and a table next to each other. The edges of the two were parallel, but slightly displaced along the perpendicular direction. As you can see in figure 4-3, the line feature detector tended to merge the two features into one long line whose actual orientation was skewed to accomodate the displacement.

This could, of course, by lowering the threshold for perpendicular displacement of merged lines (i.e., requiring them to be closer together). Unfortunately, doing so causes other problems — in particular, when the robot returns to a previously seen area with some small accumulated horizontal error, it may fail to correctly merge the feature corresponding to a wall with the previously viewed version of the same wall.

It is possible that some of these difficulties could be dealt with by using robust statistics in place of means, and variances, in determining split criteria and position corrections. Unfortunately, robust statistics (the simplest of which is the median) are very difficult to compute on an incremental basis, which is what is required to allow any two line features to be tested for fit in $O(1)$ time. (The total time for testing the merges in the test implementation was $O(n^2)$, where n is the total number of

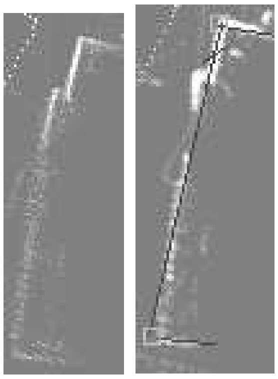


Figure 4-3: Feature detector misbehavior — two pieces of furniture give rise to features which are, by eye, adjacent, but parallel lines. However, setting the pependicular distance threshold low enough to prevent them merging prevents other legitimate merges as well, particularly when the robot is returning to a previously seen environment, as on the right.

line features; this could presumably be reduced to something more like $O(n \log n)$ by imposing a suitable search tree).

A similar problem was encountered with the corner detector. The sensitive parameter in this case was the maximum distance between the endpoints of the lines forming the corner. If this distance was set low, then legitimate corners were missed as can clearly be seen in figure 4-1. However, if it were set high enough, then lines which did *not* seem to form corner features "by eye" wound up being classified as such by the feature detector.

Other problems

One contributing factor to the unreliability of the feature detectors given above, generally, may well have been that they were based on the data which was instantaneously recovered from a single frame. As we have repeatedly seen, the obstacle points which accumulate on the short-range maps have a considerable spread around the locations corresponding to physical obstacles, due to errors in camera calibration (and lens effects which we have not modeled). In the final short-range maps, we can deal with this by using thresholding or non-maximum suppression to restrict the raw data to individual obstacle points which have been repeatedly sighted and are likely to be valid. (As we shall see below, thresholding is adequate in practice). However, if primitive features are detected on the basis of data retrieved from an individual frame, this is much more difficult to arrange.

In particular, the above effects made it very awkward to try to use the robot's visually detected features over successive frames to augment dead reckoning. Since the same features are visible from frame to frame, it is an attractive idea to use those features to correct the odometry. However, this involves features based on very small numbers of data points (when the robot is exploring); in that context, errors introduced by camera calibration and the discretization of points to pixel coordinates adds up quickly. A Kalman-filter approach to incorporating these corrections would probably have helped considerably. However, this in combination with the errors resulting from the mismatch of the feature detectors to the environment, suggested it might be better to attempt to use a different kind of feature.

4.4 Simple place-representations

To summarize the problems encountered with the feature detectors outlined above, they tried to be too specific about too much. For example, consider the real-world objects on which the feature detectors performed poorly — the office-chair bases and the cylindrical trash can. In each case, the robot could reliably detect a cloud of obstacle points in a consistent direction, but the feature detectors were unable to forumulate a robust and consistent response to that cloud, with the result that the robot was unable to make effective use of it.

This suggests that it might be desirable to use the generated obstacle points *themselves* as the key features for place recognition. Naturally, this can't be done one obstacle-feature at a time — as far as our robot is concerned, all individual obstacle points are exactly the same (all it knows about them is that they are not the floor, as they are detected on that basis).

If all individual obstacle points are the same, from the robot's point of view, then the only way it can distinguish two different places is by the configuration of obstacle points visible from a particular position. Accordingly, we chose to make those combinations the actual features used to recognize an individual place in our final system.

4.4.1 Rosettes

The trouble with using combinations of features, of course, is that a combination of a small number of features — say, a pair of observed obstacles — is still not very specific. For example, if the robot is near a long wall, any feature-pair which represented a pair of points, or triplet of collinear points, would match. On the other hand, if we were to indiscriminately form combinations of large numbers of obstacle points visible (after the robot were operating in a given area for a substantial length of time, and had accuulated a large number of snapshots) would subject us to combinatorial blowup.

Furthermore, since the detected obstacle points tend to cluster very tightly in locations which correspond (at least roughly) to physical obstacles, the raw detected-obstacle data is redundant. As indicated above, we can reduce this redundancy somewhat by omitting obstacle points which did not receive support on multiple frames (as outliers due to sensor noise or calibration errors), but much will remain — in particular, the redundancy of the individual collections of points along a wall.

All of this suggests that it ought to be possible to compute a small signature for short-range maps which encodes enough information about a given map to reliably determine, given another map, whether the same obstacles are in view in the second map as the first.

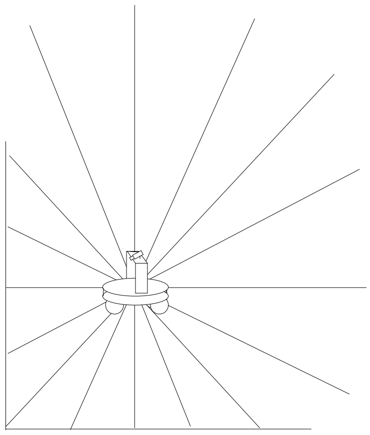


Figure 4-4: Simple example of a rosette data structure — the rosette itself consists simply of an array containing the length of the rays, as indicated.

What we have chosen to use as the basis of our signatures is the shape of the free space centered at the robot's current location — that is, the shape of the patch of floor which the robot occupies, and which is bounded by the obstacles which are immediately in sight, or by the effective range of the robot's vision if no obstacle is visible in a given direction.

The data structure which we have chosen to use as a signature is one which we refer to as a "compass rosette", which consists, simply of the distance from the robot's current position in the short-range map to either the nearest visible obstacle, or the horizon, in each of 32 directions — the name comes from the 32 points of the traditional compass. (Note that this assumes that all the nearby obstacles are, in fact, represented on the map — strategies for assuring this are described in chapter 5).

The justification for the choice of the compass rosette data structure is simple. If two places differ in the placement of the obstacles closest to the robot, then the lengths of the corresponding rays will differ as well, so long as the sampling of rays is sufficiently dense that there is a ray pointing to the obstacle in question. Objects in our environments tend to be large enough to make this very likely — at five feet from the robot, a distance which is routinely approached, an obstacle would have to be well under a foot in diameter to be likely to escape notice.

4.4.2 Properties of rosettes

A few properties of the rosette data structure are immediately apparent. Rotating the map simply rotates the order of the rays in the rosette array, and is therefore easy to deal with. On the other hand, translation of the map (or equivalently, of the robot's estimated position in the map) is more difficult to accomodate.

For example, suppose we think that the robot may be displaced by an amount (dx, dy) from the position it occupied when the rosette was originally scanned. It is possible to compute the points corresponding to the endpoints of the rosette rays after this translation. However, the result of this process is *not* the rosette we would

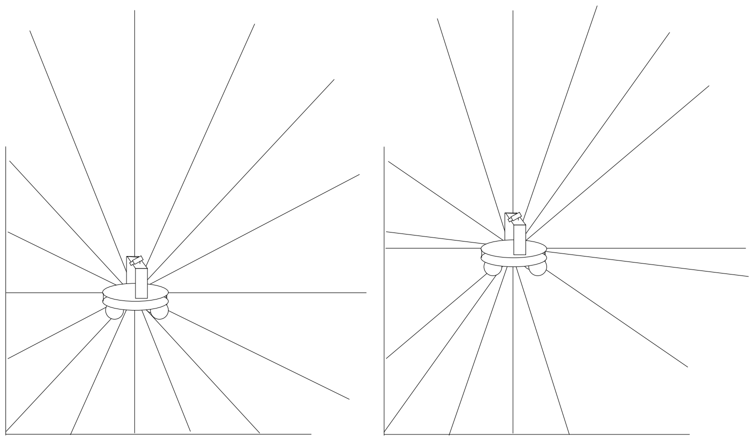


Figure 4-5: Computing the translation of a rosette is difficult to do directly, in practice, since the endpoints of the original rosette, after translation, are no longer directly in the directions of the compass points.

expect to scan in at the location which we believe the robot to currently occupy. The problem is that translation changes not only the distances to the endpoints of the rays, but the directions as well.

A consequence of this is that although we might wish to compare maps by directly comparing the rosettes which were built from them, it is not proper to do so unless we have prior reason to believe that they were gathered from places which were quite near to each other. However, we are most interested in performing these comparisons precisely when our information about the relative position of the spots at which the two maps are gathered is unreliable at best. (The whole point of making the comparison in the first place is, in practice, to try to estimate that displacement, or to determine if the maps in question are so mismatched that they cannot represent the same place). We need another procedure.

4.4.3 Matching rosettes

We have seen that we cannot reliably compare one scene (or robot location within a scene) to another simply by comparing the corresponding rosettes. However, recall that we are primarily interested in place representations as a way of determining, when the robot arrives at some location, whether it has been near there before, and if so, then where exactly with regard to its former location.

For such comparisons, we always have the sensory data corresponding to the current location available, so we can simply match the rosette corresponding to the previously viewed location to that.

Seen in this light, the problem we actually face can be viewed as finding the best match of a model (specifically, a model of the bounds on the free space surrounding the robot, as implemented by the rosettes) to the short range map, which can be viewed as a 2-d image of the immediately surrounding space.

Matching: Basic definitions

To be more specific, adopting the language of computer vision, we seek to match a group of *model points* — the points comprising the visible obstacles at the endpoints of the rays of the rosette — to a group of *image points*. Choosing the match may involve rotating or translating the rosette (since the robot's estimates of its position and orientation may both have drifted); it does not, however, involve scaling it, or performing any sort of 3-d rotation. (This follows directly from our key assumption that all obstacles lie on the floor — our navigation algorithms believe that they live in an essentially planar world). Boiling that all down:

> The problem of finding a match is *defined* as the problem of finding the translation and rotation which brings a rosette into closest alignment with the current short-range map.

It is worth noting, for future reference, that any of the rigid 2-d transformations which define a match can itself be determined by three numbers — the x and y displacements of the translation, and the angle θ of the rotation. These three numbers can be viewed as the coordinates of the match they define in a space of possible matches. That space is referred to as *transformation space* in the ensuing discussion, and in the computer vision literature.

In what follows, we will denote the rosette which we are trying to match as R, and say it consists of individual points denoted r_i. Likewise, we will call the map which we are trying to match it to M and say that it contains obstacle points m_j.

There is a a great deal of material on these sorts of alignment problems in the computer vision literature, which we can only review here in the most cursory manner. We concentrate here on a development of the particular alignment schema which we actually used; for a more thorough review, see [19, 42]. (Not all of this literature could be applied in any fashion to our present problem — for instance, we are doing a 2-d alignment because both our images and our models really *are* naturally 2-d, and not because we find it to be a useful approximation to a 3-d world; this makes any of the various schemes to deal with, say, perspective irrelevant). The particular scheme we used is an adaptation of the RAST algorithm due to Thomas Breuel [8]; the algorithm is itself essentially RAST, though we do add a few tricks, for efficiency, which have the effect of reducing RAST's worst-case exponential run time to a polynomial bound.

We begin our discussion of matching by discussing a simple, rough and ready algorithm which has been widely used for simple alignment problems. We then discuss some of its drawbacks, and develop a more satisfactory version.

Matching: a simple algorithm

We begin by discussing a hoary alignment technique which has been around about as long as computer vision itself, the *Hough transform*. So let us suppose, again, that we are trying to match a rosette R to a map M. That is, we are trying to determine the translation and rotation (i.e., the point in transformation space) which causes R to "best" match M, for some sense of "best" which we must now more precisely determine.

Now, in a "good" match, the individual rosette features r_i should obviously be matched in some reasonable way with the individual match features m_j. Let us call such a pairing a *correspondance* (again, following computer vision terminology). Many different transformations can give rise to any given correspondance, of course — imagine placing the point r_i physically atop m_j, and then rotating R around it. However, most transformations will not, in fact, place r_i atop m_j; they will place it elsewhere. That is, only a very few transformations are *consistent* with any given correspondance. See figure 4-6 for an illustration; the caption of that figure has an extended explanation.

(The figure illustrates the case of transformations which are simple 2-d translations. The geometry of handling rotations is more complex, but the basic idea is the same. In fact, for any given rotation, there is precisely one translation that brings r_i atop m_j — the set of consistent transformations therefore forms a helix in transformation space, which winds around a line parallel to the θ axis).

We are now ready to define the Hough transform. To perform a Hough transform, we begin by discretizing transformation space (as defined above), and constructing an array of cells corresponding to the discrete regions of transformation space, all initially containing zero. We then consider every *possible* pairing of rosette and map obstacle points. The transformations which are consistent with such a correspondance will only occupy a few of the discrete regions of transformation space. We increment the array entry for each such cell.

The assumption behind this process is that there will be many more correspondances which are consistent with a good match than with a bad one. If that is so, then at the end of this process, the counts associated with cells in transformation space which contain good matches should be high compared to the others. In fact, the simplest approach to using the Hough transform consists in simply taking the midpoint of the "winning" cell in transformation space as the best match available.

Hough transform pitfalls and drawbacks

The Hough transform is simple and relatively fast ($O(|R||M|)$ is about as fast as alignment algorithms get). So, what are the problems with it?

For our purposes, the main problem with the Hough transform is that the implicit criterion which it establishes for "best" match is not necessarily the one we want to use. In particular, matches wind up being scored on the total number of correspondances which are allowed for by the match itself, or by other matches which are nearby in transformation space. Thus, a potential match which brings five rosette features close to five map features each will be scored much better than another potential match which brings fifteen rosette features close to one map feature each, even though we would prefer for the second match to be preferred, as it actually accounts for more of the rosette features.

Another problem with the Hough transform, in many applications, is that it deals more poorly than it might with noise in the position of rosette and map features. Because there is noise in both sets of positions, there will in practice be some degree of error associated with the correspondances. So, as shown in figure 4-6, the region of

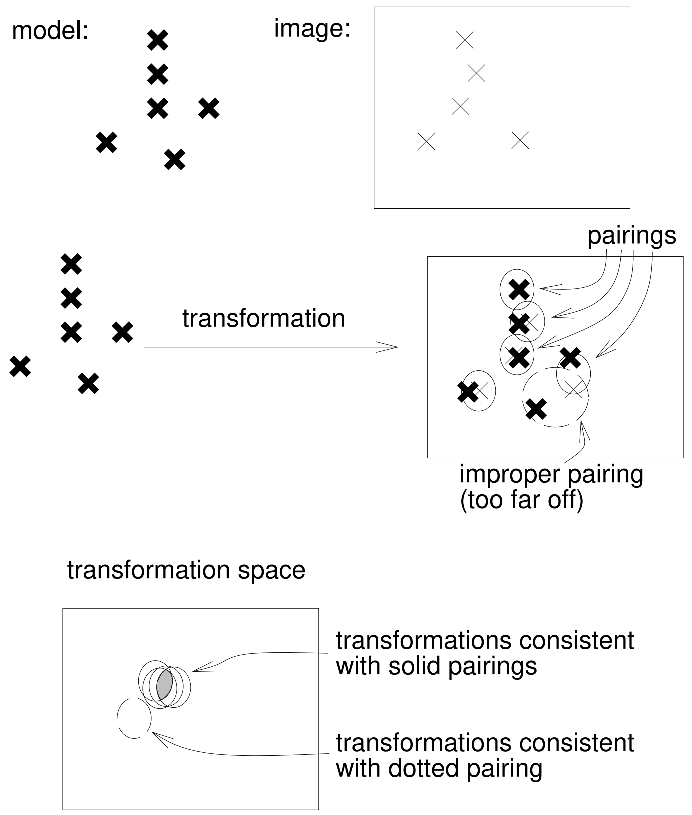


Figure 4-6: Matches consistent with a given correspondance. Consider a model, as at top left, and an image as at top right. If we translate the model to a particular point on the image, and look to see which model features are within a given range of an image feature, we arrive at a particular set of possible pairings for the translation, which may leave some features (from either set) unpaired.

Now, if we had translated the model to a very slightly different location, the same set of pairings would have been possible; however, if we translated it too far off, they would cease to be possible within the error criterion (and other individual feature pairings, such as the one indicated by dotted lines, might become plausible). Thus, each *pairing* can be seen as consistent with a particular set of *transformations*, as indicated at the bottom, in a diagram which also shows the set of transformations consistent with the dotted pairing (which is *not* consistent with any of the others shown). The set of transformations consistent with all pairings shown is limned in grey.

Since any superimposition of the model on the image corresponds to some transformation, the RAST algorithm looks for "good" superimpositions by trying to find "good" regions of this transformation space.

transformations which is consistent with any given correspondance should really be properly viewed as a three-d form whose axis is the set of matches which are exactly consistent. The Hough transform accounts for this to an extent by lumping together matches that form in the same discrete region of transformation space. However, it will not similarly lump together matches in adjacent regions of transformation space, no matter how close they both are to the boundary. Thus, potential matches which are close to those boundaries are effectively at a disadvantage.

For our particular application, this is less of an issue than it would be in others, because of the peculiar nature of the maps with which we are aligning the rosettes. In particular, we have observed that the short-range maps generally have a band of detected obstacle features clustered around the actual, or at any rate likely, positions of the obstacles (as viewed from a given point). Furthermore, as this "cloud" grows larger, we become less certain about the actual position (having more inconsistent reports to go on).

We can therefore use the noise reflected map itself as a first-order estimate of the noise in the mapping process. That is, instead of looking for inexact matches to the estimated position of an obstacle points, we look for an exact match to a member of the cloud of detected obstacle points surrounding it. In other words, we are able to make use of methods which impose hard boundaries in transformation space because the mapping process itself "smears" related transformations across the boundaries.

A faster algorithm

We are now nearly ready to specify our final algorithm. It is easier to do so if we take rotation out of the picture for a moment, by assuming a fixed rotational position for the rosette. (We will allow it to rotate again shortly). Since the orientation of the rosette is fixed, the only variable part of a match is now the translation. Since each possible correspondance of rosette and map points gives rise to only one translation, it now gives rise to only one possible match — that is, to a single point in a 2-d transformation space consisting solely of two-dimensional translations.

Now, any reasonable standard for a "good" match will require it to correspond to a cluster of these points in transformation space, although we will want to know more about the cluster than simply the number of points containied in it (which is all we could get out of a Hough transform). Still, to find a technique for identifying potential good matches, we may be able to start with some sort of clustering algorithm to the set of transformations, and adjust it to judge potential high-volume clusters on some basis more sophisticated than the raw density of the points in the clusters.

Our algorithm begins as an adaptation of a simple and effective clustering technique. It works by repeatedly choosing a dimension along which to partition the space (say, x), and a threshold along that dimension (say, 3), and then dividing the space into two portions, divided by the hyperplane determined by the two, thus dividing the space into two cells (say, $x < 3$ and $x > 3$). Each of these cells may then by subdivided in the same manner, and the whole process may be continued as long as there are points in the data set being clustered to support the split criterion. By repeating this procedure, choosing a new dimension and threshold each time, a recursive structure is induced on the space being clustered.

This procedure of multiple splits must cease, of course, at some point. If the point of the procedure is simply to do clustering, the easiest thing to do is simply to cease subdividing when the number of raw data points in the partition to be split is less than a certain amount. The complete tree that results is then an adaptive subdivision of transformation space (the AST of the RAST acronym) according to the density of the data, fine-grained in regions which contain a large number of data points (i.e., possible feature correspondances), coarse in regions where data is sparse.

One drawback to this particular form of subdivision is that while it is broadly adaptive, the actual boundaries between partitions are generally not well-fit to "natural" boundaries of the data (i.e., "form-fit" to the actual shape of a cluster). It is therefore possible, in general, that a cluster of feature-matches sufficient to sustain a match will be split between two cells by an infelicitous poor choice of boundary plane. Indeed, the worst-case exponential behavior of the full RAST algorithm is due to the need to compensate for such effects.

However, the reason such a cluster might spread across the boundary between partitions in the first place is due to noise in the imaging process. This is where the "smearing" discussed above helps us — if there is a true match between model and image features within error bounds, then there will be a match to a "smeared" image point with works exactly. Therefore, it is an acceptable approximation to simply keep on partitioning the space into smaller partitions, until the partition size is less than the size of the expected "fuzz" — at which point, the partition-induced boundaries will not be contributing significantly to the actual error in the match. If the feature matches left in the partition at this point are sufficient to sustain a model match, this then becomes a candidate alignment of the model to the data.

We may schematize this general form of algorithm as follows, generalizing the task slightly to be the task of finding a match better than some preexisting other match:

```
function find_best_transformation (pairings, region, best_match) =
   let best_possible_score_in_region = score_best_possible_match (pairings)
   in

   if (best_possible_score_in_region < score_of_match (best_match))
      return best_match
   else if (region_size (region) < min_match_size)
      return make_match (centroid (region), best_possible_score_in_region)
   else
      let (reg1, reg2) = split_region (region)
          points1 = points_in_region (pairings, region1)
          points2 = points_in_region (pairings, region2)
          best1 = find_best_transformation (points1, reg1, best_match)
          best2 = find_best_transformation (points2, reg2, best_match)
      in
          if (score_of_match (best1) > score_of_match (best2))
              return best1
```

```
        else
            return best2
    end
end
```

In this case, the regions are rectangles (regions of a 2-d transformation space), which are split simply by halving them along the long dimension (the treatment of rotation being discussed later). The main issue left to be decided is the nature of the function `best_possible_match_in_region`, which determines for a given collection of points — that is, of matchings of model features to image features — the score of the best possible *complete* model-to-image match. This is, of course, the next subject for us to discuss.

Match criteria Now, we wish to use this sort of algorithm in order to find tightly bound regions of transformation space containing a set of raw data points — in our case, a set of transformations pairing a specific model feature to some specific image feature — to sustain a "good" match, by some criterion. As we saw above, one of the major deficiencies of the Hough transform is that it allows multiple matches for a small set of feature points, thereby allowing a region of transformation to be scored as a "match" if a small number of model features are matched, say, five times over each, even though many other model features are unrepresented.

Indeed, as noted above, this effect would be particularly pronounced in our case as we are deliberately counting on a large and noisy set of potential match points to compensate for noise in positional estimates. Counting the raw number of possible matches is likely to give severely misleading results in this case, at least as reflective of the amount of imaging noise as they would be reflective of the fit to the data (as the more noise there would be, to a certain point, the wider the band of "fuzz" around the centroids of the data, the more image points to form potential matches, and hence, the more total potential matches in total).

So, we wish to specify a match criterion which avoids this difficulty. The most direct approach is to require the region containing the possible match (subject to further splits) to contain matches for at least a set number of model points, as opposed to counting the number of raw matches.

However, it is quite likely, given the nature of our data, that if there is a transformation which includes pairings for n model points in a given region of transformation space, then there will be others with the same number of matches nearby, since the image features being matched have been "smeared" as discussed earlier. We would, of course, like some principled way of distinguishing among these. To do this, we observe that in our original data, not all image points are created equal — some will have been darkened in the short-range mapping process over several individual scans, and there is therefore better evidence for an obstacle at these image points than for the more marginal ones. We therefore prefer matches which have pairings to less marginal points. That is, given two otherwise equal matches (same number of model points paired), we prefer the match whose *worst* paired image feature is the *least bad*.

To summarize, we evaluate the best possible match which may be achieeved by choosing from among a set of pairings, we may score it as follows:

```
function score_best_possible_match (pairings) =

    let feature_present = array[num_rosette_rays] of boolean
        worst_feature_yet = dark;
        number_rays_matched = 0

    for i := 0 to num_rosette_rays
        feature_present[i] := false

    for pairing in pairings
        feature_present[ image_pt_idx [pairing]] := true
        if (image_pt_darkness [pairing] < worst_feature_yet)
           worst_feature_yet = image_pt_darkness [pairing]

    for i := 0 to num_rosette_rays
        if (feature_present[i])
           number_rays_matched := number_rays_matched + 1

    return (number_rays_matched * max_darkness + worst_feature_yet)
```

Dealing with rotation We now must return to the question of rotation. Unfortunately, building data structures to represent transformation spaces which include the effects of rotation directly makes accounting for potential matches a great deal more difficult to manage computationally, as any individual feature match no longer induces a single point transformation in a 2-d transformation space, but rather a helical set of potential transformations in a 3-d transformation space (x, y, θ).

An alternative which has been adopted [8, 10] is to simply quantize the rotation space, and perform a purely translational matching on each of several rotated models, choosing the best. This strategy is particularly easy for us to adopt here, since the models themselves have a natural point of rotation (around the centers of the rosettes). Note that this easily allows us to limit the range of allowable rotations of the model relative to the image, simply by using a less-than-complete set of rotated models. This is useful for two reasons — it bars some otherwise likely-seeming matches which would be erroneous (see figure 4-7) and it causes the match to be found in less time overall.

This means that we deal with rotation by invoking the `find_best_transform` routine defined above through a front end which handles rotation for it. The algorithm is as follows:

```
function find_best_match (model, features, rotations, threshold) =

   var best_so_far = make_match (Impossible_Translation,
```

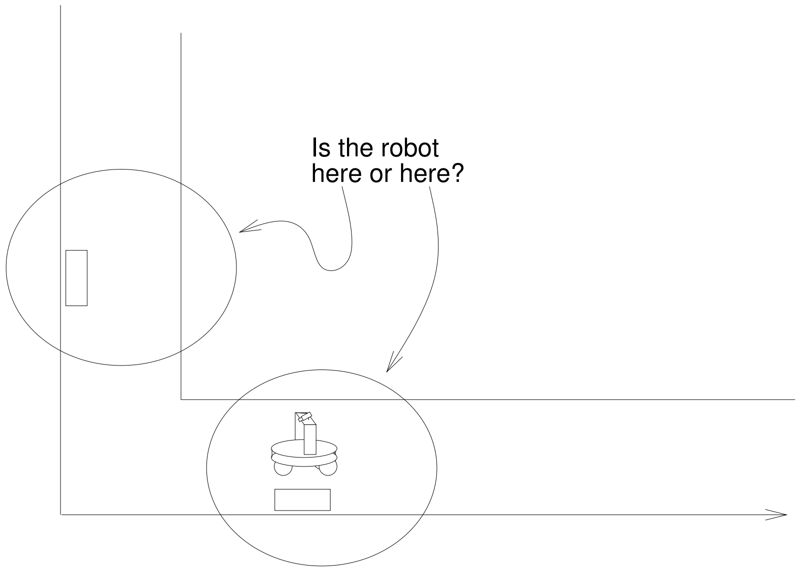


Figure 4-7: An unreasonable potential match blocked by limiting the relative rotation between image and model — the corridors on each side of the corner look pretty much alike to the robot, but the amount of relative rotation between the two makes an actual match unlikely.

```
                                threshold)

  for (rotation in rotations) do
      let this_model = rotate_model (model, rotations)
          pairings = make_pairings (this_model, features)
      in
          best_so_far := find_best_transformation (pairings, Max_Space,
                                                    best_so_far)
      end

  return best_so_far
```

This completes the basic explanation of the matching algorithm.

4.5 Results

The matcher performs quite well given the data available to it. However, the data is not always adequate for completely sound judgment.

For instance, in an environment where there are large amounts of corridor space, it will turn out that all the corridors look alike. The illustration in figure 4-7 illustrates one particularly awkward way in which things can go wrong — in this instance, the two pieces of corridor on either side of a corner look identical to each other. This source of error can be eliminated simply by constraining the allowed amount of rotation between the model and the corresponding set of image features;

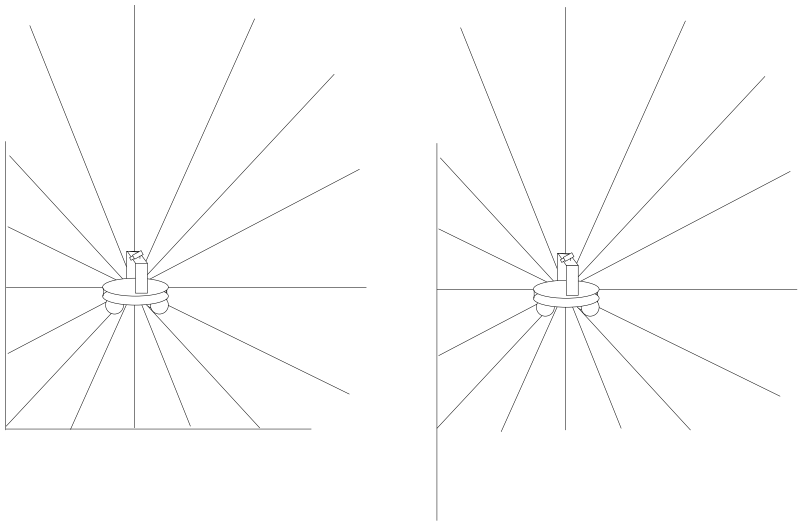


Figure 4-8: Alignment succeeds — but is this a good match?

indeed, constraining it to be less than 45 degrees eliminates the error shown. However, that restriction comes at a price of reduced robustness; if the robot base has been maladjusted (if the wheels are siginificantly out of alignment, for example), then an accumulated error in heading of 45 degrees is not out of the question, even in a fairly short run of straight-line operation, due to precession of the wheels.

Another sensitive choice in use of the matcher is the criteria for accepting a match. The most obvious choice, given discussion above, is to simply threshold the match quality function — in particular, to require a minimum number of rays of the rosette to match obstacles in the scene. However, pathological situations are encountered here as well. For instance, consider the situation in figure 4-8. In this case, alignment has succeeded, and it certainly is the best fit of the given model to the given image data. However, the match itself just "doesn't look right".

We might attempt to deal with this by simply increasing the threshold. However, empirical experience, even in simulated environments, which one may expect to be unusually well-behaved, is that raising the threshold high enough to exclude this match raises it high enough to cause false negatives — that is, the failure to recognize a place which the robot has, in fact, occupied in the past, with consequences discussed in the next chapter.

The best workaround we have found for this is to condition acceptance of a match additionally on the size of the largest *gap* in the matched rosette rays (excluding, of course, those rays for which the closest obstacle, if any, in the target direction was over the horizon). Again, this has potential problems, in particular, for adaptation to new environments — in particular, by making the system more sensitive than it would be otherwise to the motion of an obstacle.

That, in turn, could, once again, cause missed matches, causing unnecessary triggering of map-repair mechanisms.

One final consequence of the sparsity of some rosette place representations is that some rosettes are difficult to use properly for the recognition that the robot has

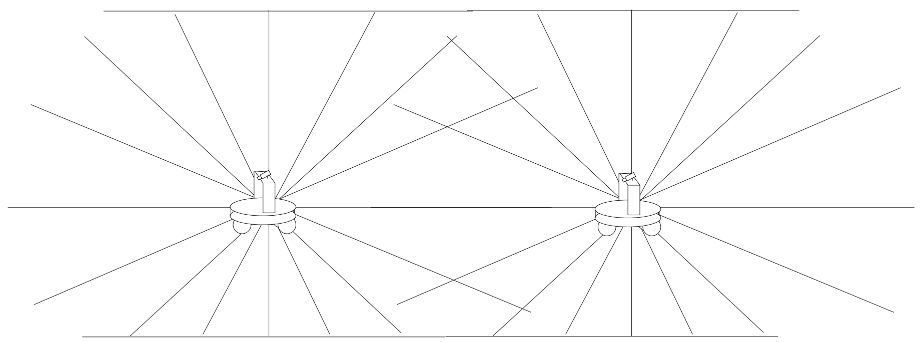


Figure 4-9: Identical place representations — how much can we tell about where we are?

arrived at a particular place and not one of its neighbors. The canonical case, once again, is place representations made in long corridors. The problem with these is that the representation of any given stretch of corridor is pretty much identical to the representation of any other stretch of the same corridor. (See figure 4-9).

As can be seen from the figure, it is difficult to use any of the place representations from inside the corridor to get a definitive position fix — the robot can tell from the alignment to the corridor how far displaced it is from the centerline, but not how far it has been displaced along the axis of the corridor. (Indeed, the situation is even worse in place representations built in large open spaces, in which the only feature visible is often a single wall).

It is useful to be able to determine whether a place representation is "loose" in this manner, or whether it does allow for a complete position fix. It is, in fact, easy to construct such a measure, given the following observation: the problematic place representations discussed above are problematic because the configurations of obstacles represented along the rosettes are all along one, or at most a few, collinear lines. Place representations which *can* support an exact position fix are likewise able to support an exact fix because they have obstacles along multiple lines. (See figure 4-10).

So, what we would like is some way of testing whether the obstacles are "too collinear". Now, the obstacles in a rosette representation have no particular directionality associated with them. However, if we draw lines connecting the endpoints of adjacent rosette leaves, as in figure 4-10, those lines do, and it makes sense to ask whether they are all parallel, or not. In fact, our "goodness" criterion is a bit more strict, to avoid a single outlier marking a rosette as fully supporting recognition because a *single* one of these lines is perpendicular to all the others, which are parallel in a body. What we require is that there be a minimum number of lines which are themselves perpendicular to several other lines from the set. Rosettes which are under threshold either way are marked as not supporting completely reliable alignment.

Place representations with this unfortunate characteristic are treated by the actual map-building machinery (discussed in the next chapter) with a certain amount of unusual care, to attempt to deal with this problem. In particular, the robot's machinery for recognizing that the robot has gone in a circle, and arrived at a heretofore

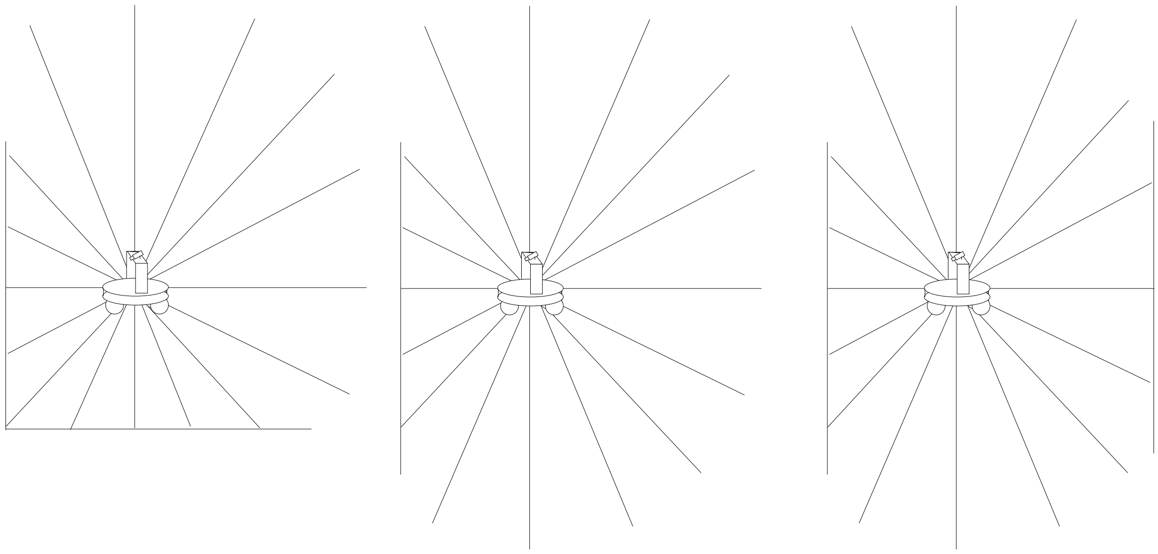


Figure 4-10: Place representations which do, and don't, support exact position fixes — the better case on the left. Displacement between endpoints of adjacent rosette leaves, represented as vectors, are shown for each. Note that they are all more or less parallel for the bad case, and that each vector has several others at a considerable angle from it, in the good case.

known place by a previously unknown route, requires additional evidence before recognizing a place which does not fully support alignment, compared to the evidence it requires for a place representation which does.

4.6 Conclusions

This chapter has discussed an approach to extracting data from a short-range map which is of use in matching it to future short-range observations (and also discussed some alternative approaches which did not work so well in our application). Our approach is to keep reasonably close to the raw data, so long as it can be made to server, as deductions made from it may be made inaccurately.

However, this technique, by itself, cannot serve as a map. These rosettes must be combined somehow into a larger structure in order to be actually useful as a map. That structure is what we now discuss.

Chapter 5

Building the Long-Range Map

The preceding chapter concentrated on building representations of the landmarks which the robot can perceive in a given location, and using those representations to determine whether the robot has previously seen its current location, and if so, correcting its position estimate.

This chapter is about how to use that information to build up a structure which contains global information about the robot's environment. We begin by briefly recapping the reasons for building the graph structure, and the problems it has to subserve. We then discuss the structure itself, and the algorithms for maintaining and exploiting it. Finally, we show results in simulated environments.

5.1 Why are we building the graph?

To begin our discussion of the graph data structure, let us remind ourselves of the purposes it needs to serve. As outlined in chapter 4, these are:

- Storing information about a location the robot occupies
- Matching the robot's current position to other locations it *might* be occupying, and estimating the accumulated drift in the robot's dead-reckoned position estimate if it has indeed returned to a previously seen place.
- Suggesting useful directions for exploration.
- Allowing the robot to plan paths from its current, known position to some position which it has occupied in the past.
- Supporting some notion of repair in case the robot encounters unexpected conditions at a place it has already explored.

The previous chapter developed a data structure for partially accomplishing the first two goals: Given a short-range map, the rosette will store information about the immediate obstacles, but nothing about the position of that location, taken as a whole, to other places where the robot has been. Likewise, with regard to the second goal, the robot can determine given a rosette, how good a fit that rosette constitutes to the robot's current position, but the question still remains as to *which* places the robot might have returned to — i.e., which ones to check.

So, what remains of the goals above is as follows:

- Storing information about the relationships between neighboring and nearby places.
- Suggesting where the robot *might* be, if it is exploring new territory and might have returned to a previously seen location (i.e., suggesting possible circular transits).
- Suggesting useful directions for exploration.
- Allowing the robot to plan paths from its current, known position to some position which it has occupied in the past.

In short, we need a data structure which represents the facts about the local environment which the robot needs to construct and represent strategies for moving around within it.

5.2 Structure of the representation

As seen in chapter 2, the most frequently used structures for these sorts of structures are, respectively,

- Maps of the sort we have called "short-range", which represent characteristics of the space surrounding the robot, with no attempt to represent the landmarks themselves.
- Data structures which represent landmarks or obstacles surrounding the robot, and estimates of their positions within a larger space (and the uncertainties of those position estimates).
- Graphs, which may represent metric information about local place relationships, or simply the topology of the space.
- Visibility graphs, which divide space into cells according to the relative positions of nearby obstacles. These represent reliable aspects of local space without getting overly specific about geometry, but rely on being able to identify the same landmarks from a variety of angles.

5.2.1 Representation requirements

Now, let us consider these data structures in the light of the tasks delimited above, and the functional requirements and capabilities we wish to assume for our robot. To briefly recall the main considerations from chapter 2:

Reliable visual recognition of completely arbitrary visual objects is still neither easy nor cheap. This will limit the effectiveness of any strategy which requires reliable identification of particular objects — in particular aspect graphs and pure topological representations (which lack explicit or implicit metric information) both rely solely on the identification of particular objects, for use as landmarks, as their sole means of place identification. We seek a more robust approach.

On the other hand, we also wish to be tolerant of deviations in the robot's notions of its own location in space as it traverses large regions. The main problem here, to restate the issue, is that as the robot traverses extended regions of space, small deviations in its position estimates can add up to the point that, when the robot actually has gone in a circle, its internal estimates may indicate that it has followed some different course entirely. At that point, the best way to reconcile the data available with the coordinates of the obstacles in some fixed, 2-d, coordinate system is not obvious. However, it is not any more obvious that the robot actually *needs* a completely consistent global view, so long as there is a structure which represents what it has determined about each local situation. Accordingly, we prefer to eschew representations which represent all information in terms of estimated positions in a postulated global reference frame.

Of the data structures which we have considered, the one which seems to match our requirements best is therefore a graph in which the nodes represent representations of places the robot has traversed, while the links represent relationships between neighboring places, and carry some of the local metric information. Since we *do* wish to represent local metric information — it is reproducible even in situations where constructing a globally consistent metric is difficult or impossible — it is reasonable to associate a *local reference frame* with each node, and to express the metric information associated with each link in the reference frames of the attached nodes.

Of course, this begs the question of which neighbor relationships are explicitly represented in the graph. The safest policy is to explicitly represent only information for which the robot has direct evidence, and where it proves necessary to make inferences from the available data, to construct those inferences on the fly. (As we shall see, the computations which perform such inferences are generally quite quick). In particular, this consideration suggests that the links in the graph should be confined to links between representations of places that the robot has transitioned between directly, carrying information about the observed distance and bearing of the two places with respect to one another (in reference coordinate systems fixed to the sensory data).

Since the purpose of the rosette data structure developed in chapter 4 is precisely to encode the perceptually available data about the obstacles visible from a particular point, it is the natural starting point for building a node in such a graph. It is to be expected that we will want to have other information associated with the rosette itself in the nodes of such a graph; in our final system, there is additional data to support the planning and exploration algorithms which the robot has to perform.

The data structure which emerges from all of these considerations is a graph whose nodes are representations of individual places, and whose links represent paths between physically adjacent places which the robot has directly traversed. However, this choice leaves many things unspecified — most notably, when to create nodes (representing places) and links (representing neighborhood relationships between them), and what information (beyond that inherent in the graph itself) to associate with either nodes or links.

5.3 Specifications of the representation

To figure out what information to associate with various components of the graph, and how to build it, we need to reexamine what information is available to our robot, and what problems it needs to contend with — in particular, what it can and cannot know about its own position and environment.

5.3.1 The basic idea: What is in the nodes and links

We begin by summarizing the choices already made in the light of the considerations of the preceding sections, and what sort of a data structure they yield.

We are building a graph in which each node is associated with a rosette in the form of chapter 4, and a set of links between them. The simplest way to proceed is for each link to be associated with exactly two nodes, and to carry the estimated displacement (distance and direction between them). Operationally, each node in the graph would be a representation of a single place (though the robot may or may not be able to obtain a position fix relative to that representation when it is near the place represented), and each link would represents a known physical relationship between the two places, each reachable from the other by a straight-line traverse.

We now consider a number of problems faced by the robot, and how they may be dealt with in the context of such a structure.

5.3.2 Problems faced by the robot

Local — Can we get a position fix?

To begin with, the visual data available to the robot in a location (or at least those data which are explicitly represented in our rosette data structures) may not permit the robot to determine its position relative to the center of the place represented. For instance, consider the three place representations in figure 5-1. If the environment is sufficiently rich (for instance, if there is a visible corner, as indicated), then the robot can use the data it has to gain a position fix. On the other hand, if all the robot can see are lines which all in parallel — as, for example, if it is in a long corridor or approaching the wall of an otherwise empty room — it can only determine its distance from the wall (or the walls of the corridor), and not how far *along* the wall or corridor it might have travelled.

We might want the robot to simply attempt to avoid these situations. Unfortunately, this is not always possible. For example, suppose the robot is in the middle of an open area. If it is in the center, and moves out towards the sides, it could easily find itself in a situation in which it is blocked, and the only visible feature is the wall that is blocking it. At this point, it can neither continue in the direction it is going (it is, after all, blocked), nor can it form a place representation which is capable of sustaining a position fix, as the only visible feature is the single wall blocking it.

Another problem with this notion is that the robot may, in some environments, have to follow somewhat complicated paths which do not, at any single point, have

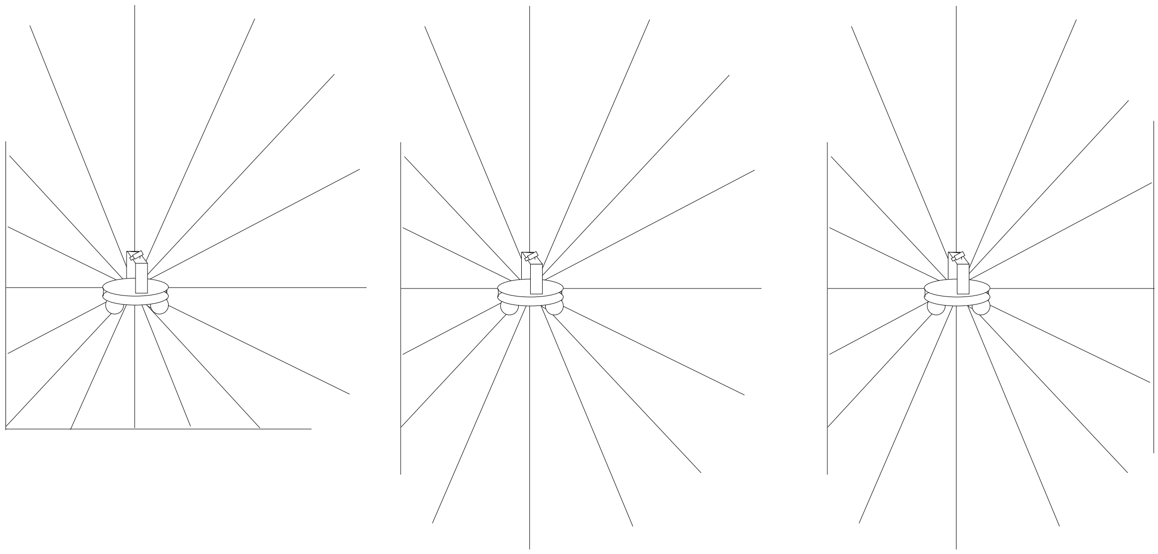


Figure 5-1: In a place with relative obstacles as indicated on the left, the robot can determine its 2-d position relative to the obstacles uniquely from range observations. However, in the other two cases, it can only determine its position along the corridor or wall.

enough features to sustain a visual position fix. For example, consider a long, gently curving corridor. One stretch of the corridor looks pretty much like another, so as with a long, straight corridor, the robot cannot determine its position along the corridor on the basis of visual range data alone, so as with a single, straight corridor, we might think of representing it as a single link.

However, if we tried to represent a curved corridor by a single link, the robot would have to be able to use that link in order to determine how to traverse the corridor. (Remember that storing information about how to get between the places represented is the main point of building the map). Coming up with a representation of such complicated motor programs would load a great deal of data onto the links (a potentially indeterminate amount — consider an indefinitely long sinusoidal corridor). Furthermore, it would not necessarily preserve the desired invariant that every node in the graph can sustain a full position correction, since (as we have seen) that is not necessarily true of the *ends* of the corridor.

To complicate matters still further, we would also have to come up with motor annotations for links which are created not as the result of an immediate motor action of the robot itself, but rather in the process of reconciling duplicate descriptions of a portion of the environment (created in situations where the robot cannot avoid it; see section 5.4.4).

In addition to the considerations listed above, there is one other purely practical matter, concerning not the needs of the mapping system itself, but rather applications of it. It may be the case that there are several distinct locations along an otherwise featureless corridor which the robot must know to treat differently because of the application for which it is being put to use — closed doorways of distinct offices, for example. Creating several graph nodes representing explicit positions along the corridor gives us a convenient "hook" in the map representation itself for task-related information of this sort, regardless of its nature.

Since, unfortunately, we will have to put some nodes in the graph which don't have enough information to sustain a position fix, we can at least take advantage of that situation to create an alternative representation of the "corridor-like" situations discussed above — as a series of these nodes, connected by short straight-line paths, which approximate the (e.g., curved) path required to traverse the corridor. This representation has the added advantage of storing sensory information about intermediate positions in the corridor, which may at least be useful to correct the robot's position in one direction, if not in two (e.g., across the corridor, if not along it).

But, this brings up another problem — as discussed above, the robot's odometric data is unreliable; error piles up, and can do so fairly quickly. Accordingly, we need to decide when we can and cannot make use of it. (This is problem which we did not consider in the previous chapter, since that chapter was concerned with representing a single place, and odometric data is only useful in considering the relationships between places).

Global — How far can we trust odometry?

This brings up the interesting issue of the proper use of odometric data, and once again we have the same sort of dilemna as posed by place representations which cannot sustain a position fix — on the one hand, it is desirable to avoid them where possible, while on the other hand, there are situations where the alternatives are either worse or nonexistant.

We have already discussed one situation in which there is no useful alternative to use of odometric data — long corridors, in which odometric data is the only way the robot has to determine how far along the corridor it has advanced. There is another, equally important situation in which odometric data provides the only useful basis on which to make an important decision.

Consider the case illustrated by figure 5-2. This figure shows two possible paths for a robot in two similar environments. In one of these cases, the robot travelled in a path which has not visited any location more than once. In the other path, it has travelled in a circle, and returned to its starting point. In order to accurately build a map (of any sort) which represents its actual situation, the robot must be able to determine whether it has gone in a circle or not. However, the only way to distinguish between these two environments (and hence, between the circle and no-circle cases) is by determining the distance travelled along the bottom corridor. (In particular, the two cases cannot be distinguished by examining the local visual range data at any point along any of the paths).

Likewise, in the apparently simpler case shown in figure 5-3, there are two place representations whose local configurations of obstacles are identical. However, the robot will misrepresent its environment badly if it tries to use the same node in the graph to represent both of these places. Again, odometric data is the only way to tell the difference — after all, if there is so much odometric slip that the robot has gone

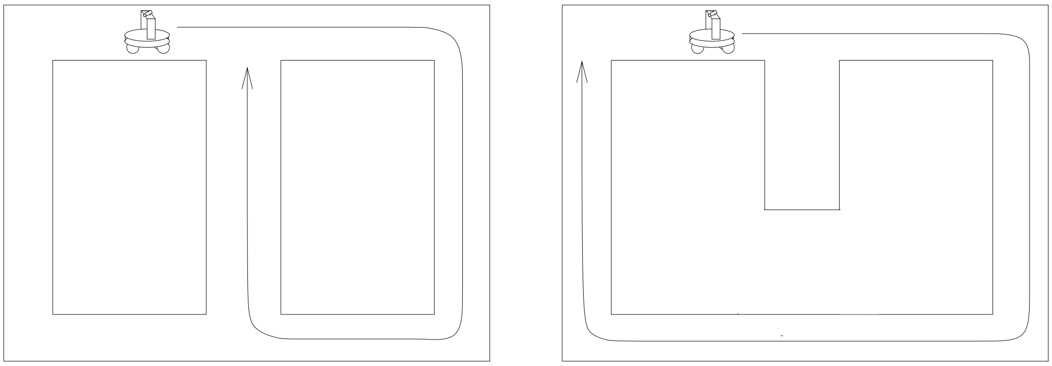


Figure 5-2: A situation in which odometric data has to be used to make a key decision. In one of these cases, the robot has gone in a circle; in the other, it has not. However, the only way to distinguish these two circumstances is by noting how far the robot has travelled along the lower corridor.

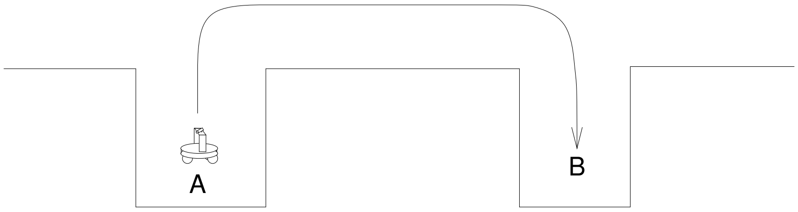


Figure 5-3: Another situation calling for the use of odometric data — the two places A and B "look" identical, but one must not be falsely recognized as the other.

almost nowhere in the "east-west" direction, it very well *might* have gone in a loop in figure 5-3.

The particular way in which this dilemna manifests itself for a graph-based mapping system is that it needs to determine whether or not to create a particular link — in the above examples, whether to create a link from the node representing its initial position to that representing its most recent position. And, as we have seen, the question of whether or not to create this link can only be answered by directly or indirectly using odometric data.

Spacing of place representations

One other question conerning place representations is worthy of note — how often, when, e.g., going down a corridor, should the robot make a new place representation? We have already decided, in effect, that the robot should stop and make a note of where it has arrived whenever it can proceed no further in its current direction of travel. However, even if the robot is not blocked, we still wish for it to have the opportunity to get a position correction, every so often, at least in terms of, e.g., its distance from the wall or corridor it is following (if not the difference along it).

As mentioned above, we could, in principle, provide for these mid-course checks one of two ways. One approach would be to augment the links in the graph representing the traverses with auxiliary information representing information about intermediate positions for use in position checks. However, as above, that would complicate the details of the structure to no obvious purpose; it is simpler to elevate the information about auxiliary positions to be full nodes in the graph.

We thus specify that the robot should form a new place representation when it has travelled from an old one either until it has blocked, or until it has travelled for a set *exploration step distance*. Now, setting this distance to be greater than the robot's effective horizon — that is, the limit of the range-detection technique of chapter 3 — is obviously fraught with risk; traversing a single graph link would carry the robot through territory of which it had no explicit representation whatever.

On the other hand, there are practical reasons for not wanting the place representations in the graph to be too close to each other either. Firstly, creating a denser coverage of the environment by place representations means having to support more of them. If we ever need to perform operations on them which take a substantial amount of time (and the alignment operations discussed in 4 are such operations), more place representations will require more of the expensive operations, and slow down processing.

Furthermore, representations for places which are only a small distance away from each other will be difficult to tell apart; this may lead to unreliability in management of the graph, particularly if the robot fails to infer a complete set of neighbor relationships between this dense set of place representations — indeed, this consideration leads directly to a discussion of our next requirement for the graph structure as a whole, maintainability.

Conditions for forming a link

It would be nice if, in cases of doubt, it were possible to establish links in the graph with a certain probability attached, so that Bayesian reasoning could be used to update the likelihoods based on subsequent data, and resolve conflicting hypotheses. However, it is difficult to figure out how to do so without adding further complication to the system.

In particular, as discussed below, our robot currently has a notion of the *single* place representation which best represents its current position. However, when leaving a place representation with probabilistic links to multiple different place representations in the direction of travel, it would need to use a probability distribution over place representations to represent the final destination — a much more complex notion of the robot's state than the current system requires. In the absence of such a notion, the presence of a link in the graph — and therefore the establishment of such a link as well — must be considered an all-or-nothing proposition.

(It is worth noting here that rats and mice appear to maintain exactly such a probabilistic representation of their current location in the CA1 and CA3 gyri of their hippocampi, and given that that capability exists, the notion of a "probabilistic graph" would convey a number of advantages. We will have more to say about this possibility in chapter 7, which discusses possible parallels between our robot navigation system and recent results concerning the navigation hardware of the abovementioned rodents).

However, so long as establishment of a link remains an all-or-nothing proposition, the robot must adopt a set of definitive rules for deciding when the robot should add a link between two place representations in the graph — that is, when we have information about the relative local displacement of two (apparently) neighboring places, that we are willing to commit to that in the graph as a basis for future reasoning and analysis. In particular, we must determine how to resolve the conflicting tensions described above — that odometry picks up errors which are too large for it to be relied upon over paths with a long length traversed (even if the *net* distance traversed is quite small, as when the robot has gone in a large circle); but there are also situations where we *must* use odometry, both on the small scale (as when the robot does not have sufficient visual data for position fixes) and on the large (the aforementioned circle case).

To begin with, when the robot has actually made a single, short transit of the unit distance defined above, and formed some definite notion of the place representations representing its start and endpoints (the latter of which may have been newly created to represent previously unexplored space), then it is obviously justified in forming a link between the two.

However, there are other cases in which the establishment of links can be justified and is useful. In particular, as shown in figure 5-4, the robot can find itself, e.g., going down a corridor which it has previously visited while exploring, without any reliable information that it is doing so (i.e., long-range odometric information which cannot yet be verified in any other fashion is the only available cue). When it is doing so, it necessarily synthesizes new place representations to cover the unexpectedly pre-trod

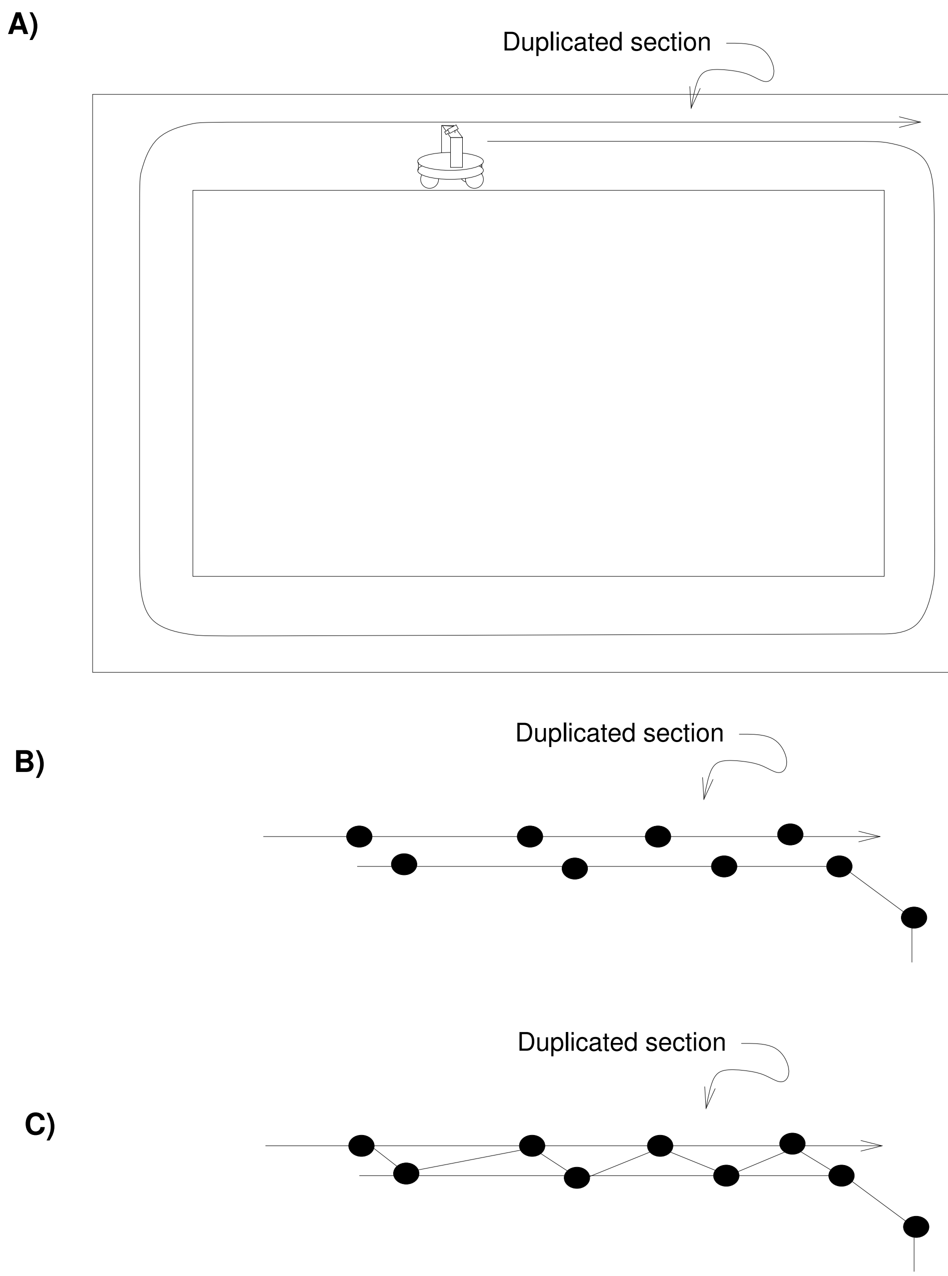


Figure 5-4: In the case shown in A), the robot finds itself going down the same corridor twice, but is unable to get a visual check as to whether it has done so until it reaches the end. As shown in B), simply making the link to the endpoints leaves two representations of other places up and down the corridor; to avoid pathological behavior in this corridor subsequently, we must add some representation of the physical relationships between the places represented by these two sets of graph nodes, as shown in C).

ground. The result of this, when the robot is finally able to corroborate the odometry by obtaining a position fix on a recognizable place representation, is multiple distinct representations of the same corridor, with no direct representation in the graph of the (now known) relationship between them.

Since the only representation we have currently for physical relationships between place representations is the link, it seems simplest to represent the (now known) relationship between the place representations comprising the two traverses of the corridor as a set of new links. It is important to note that in this case, the relative displacements of the two sets of place representations can be estimated from the visual fix, combined with the purely local odometric data recorded in the graph from the two original traverses (which, as relatively little error accumulates with small motion, is relatively trustworthy).

5.4 Algorithm

So much for data structures. We are now in a position to discuss algorithms — that is, the procedures which the robot follows in order to build and maintain the graph. We are mainly concerned here with the procedures for building and maintaining the graph. However, the main use for the graph is allowing the robot to get from one point in its environment to another (applying suitable visual corrections where appropriate to try to deal with odometric noise), which, as we shall see, is something that the robot needs to do while exploring as well. (For instance, when it has reached a dead end, it must back out to a location which has an unexplored exit before continuing to build new graph structure).

In outline, the robot's behavior takes the form of a series of transitions from one place to another, or into unexplored space, representing the beginning and end of what the robot attempts as straight-line transits; these are followed by adjustments to the graph structure, if needed. (The transits may not actually *be* in a straight-line due to the effects of the robot's underlying obstacle-avoidance procedure and the effects of slip in the robot's motor system).

On each step, the robot must determine which way to go, and how far (these transits are always of the sort represented by a single graph link, which is to say a short, straight-line transit). Having executed the transition (using its obstacle-avoidance machinery where needed), it must then determine where it arrived, and make the necessary adjustments to its graph structure. In order to do these things, it maintains certain information about its own state in addition to the graph itself. We begin by listing that state information, and then discuss each of these problems in turn.

5.4.1 What the robot knows

We first consider what information the robot maintains in addition to the graph. In addition to the graph, the robot maintains a representation of its own state, which contains the following information:

- Which node in the graph the robot is closest to
- Estimated displacement from the center of the place represented by that node (obtained by visual registration where possible, otherwise estimated by means of dead reckoning).
- The robot's current *goal*, which is either exploration (adding new nodes to the graph), or a particular place representation which the robot is attempting to gain. The robot alternates between these two conditions, exploring as long as possible, and retracing to a likely node when all further paths for exploration are either taken or blocked.

5.4.2 Deciding where to go

The robot initiates each cycle of navigation by plotting a motion, which is either into unexplored space, or retracing its steps along some known link in the graph from its present position. The case of the robot retracing its steps is simpler, so we deal with it first.

Retrace mode

When the robot is retracing, we obviously wish to move towards the place representation selected as a goal. Accordingly, the robot searches through the graph for the link which leads along the shortest path to the goal, and attempts to traverse the next link along that path from its current location (i.e., plot a course in the distance and direction indicated by that link). The time required for a simple breadth-first search in a graph of, typically, less than a thousand nodes, is negligible.

Two minor subtleties arise here. The first is what notion of "shortest path" to use — as the actual distances covered by the links are not completely uniform, the path with the fewest number of graph links may not be the path with the shortest total distance. We choose to optimize the former, reasoning that a path which requires the robot to make fewer decisions will also give it fewer opportunities to err.

The other item worthy of note is a pathological situation which can arise due to the fact that the robot's navigation is not completely reliable — the robot can be at, say, place representation A, aim for the center of place representation B, and arrive within the scope of place representation C. This can occur because the robot was diverted from its planned course by the action of the obstacle avoidance routines. It is also particularly likely when there are several representations of distinct places within a tightly confined local area — a situation which we try to avoid creating when we can (as indicated above), but which, as we shall see, sometimes cannot be avoided.

The problem is that it may be that a search for the shortest known path in the graph from the robot's final destination (point C in the previous paragraph) may lead it back to the place it started (point A). If we do nothing about this, the robot can wind up going back and forth between point A and point C indefinitely. We resolve this situation by adopting the rule that the same link cannot be traversed more than once in the same direction in any single search for a given goal. (If all paths from a given node are removed from consideration by this rule, the robot must revert to exploration).

Lastly, of course, there is the question of what to do once the robot has reached its goal. This depends on the point of seeking that goal in the first place. For instance, if the navigation system has been commanded to seek that goal by a user, the appropriate action would be to await further commands. On the other hand, if the robot chose to seek out this particular goal because exploration in other directions was blocked, the correct procedure is for the robot to resume exploration.

Exploration mode

We now turn our attention to the case where the robot is in exploration mode, starting at its current place representation. Its goal, in this circumstance, is to add to its knowledge of the local environment. It obviously cannot do so by heading in a direction in which it knows what to expect (e.g., back the way it came from). Therefore, it must choose a direction in which it does not know with certainty what it will find (but in which it has free running room, as determined by the range to the nearest obstacle in that direction), and proceed in that direction. (At the end, it will find itself either at a place which it can recognize visually — see section 5.4.4 below — or not; in the latter case, it will synthesize a new place representation, as discussed below).

What constitutes an open exit? The most obvious criterion for an exit which the robot might choose to follow is that it can see no obstacle in that direction for at least its unit exploration distance. However, just because the robot can see no obstacle in a particular direction does not mean that it can (or should) move that way. For example the particular ray may be in the direction of a gap which the robot is too small to fit through. Such gaps are obviously not possible exits.

The most straightforward rule to deal with these considerations would be to simply require a sufficiently large gap, and then split the middle. There is, however, one additional heuristic which proves to be helpful.

Consider the case where there is a very large gap — in the extreme case, that there is a single obstacle visible. The previous rule would lead to the robot believing that there was a single possible exit, directed straight away from the visible obstacle. This is inappropriate because it risks heading the robot off into a region of relatively featureless space, in which it is not likely to be able to use sensory data to obtain useful corrections for its dead reckoning.

To deal with this, we modify the rule as follows. Sufficiently narrow gaps are rejected. Gaps above that size are split down the middle, unless they exceed a second threshold, in which case we steer close to the edges of the gap.

Previously visited exits So much for identifying possible exits. Having produced a set of them, we wish to determine how to prune those that lead to territory which the robot can determine it has previously explored.

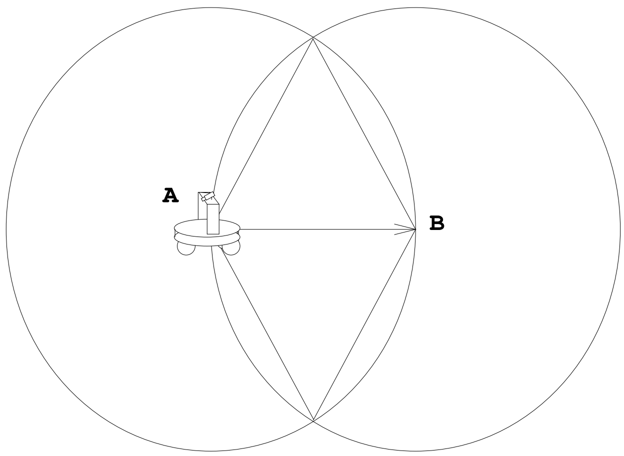


Figure 5-5: The robot is at place A, and has moved a distance of one exploration quantum from place B. It is about to move another exploration quantum — the locus of potential destinations is limited to the circle on the right. As the figure demonstrates, it will wind up in the circle covered by place B if its path is anything less than 60 degrees away from the direction from A to B — note that all lines in the figure are the same length.

The most straightfoward thing to do, given our current data structures, would be to identify the rays of a rosette with possible exits, and to declare that an exit has been explored if the robot has a representation of a neighboring place in that precise direction. However, recall that the rays of a rosette are very tightly spaced. Therefore, even if the robot has a representation of a neighboring place in a particular direction (recorded as a link to the representation of that place in the robot's place graph), it will have no *direct* representation of a place in even a very slightly different direction. Nevertheless, we want to keep the robot from going that way while exploring because, if it does, it will not in fact encounter anything new.

The preceding considerations imply that if the robot has a representation for a neighboring place in a particular direction, it should avoid exploring not only the path which is believed to lead right back to the known location, but nearby directions as well. To see how far, recall from the discussion in section 5.3.2 that the typical distance between place representations should be roughly the same as the distance the robot moves in any given exploration step. The geometry of such a situation may be seen in figure 5-5. As shown in that figure, if the robot steers less than sixty degrees away from the direction back in the direction of the neighboring place rep, it will wind up in the circle of space represented by that place, which is what we wish to avoid.

Accordingly, all potential exits within that range are regarded as "previously explored", for all *known* neighborhood relationships — that is, all those relationships which are actually in the graph. (Note that if a link is *not* explicitly present in the graph, we do allow the robot to traverse it, in order to determine if the link should, in fact, be present).

Making the final choice Taking all of the above situations into account, we arrive at a set of candidate directions which

- Does not send the robot to a previously visited location (or, more precisely, the "circle of influence" of a previously visited place representation which is *known* to be reachable from the current one).
- Does not send the robot into large, possibly vacant areas of empty space if there are other candidates available.
- Does include everything else likely to carry the robot to an area of interest.

The current choice among the remaining candidates is essentially random; we assume that whichever one we choose, the robot will eventually return, and so the current choice does not affect our long-term goal of achieving complete coverage.

And if there's no choice ...

Lastly, it is possible that the robot will find itself exploring at a place representation which has *no* open directions to choose from — that is, the entire neighborhood is either blocked or known territory. For instance, it may have gone all the way down a dead-end corridor. In order to continue exploring the totality of its environment, the robot will have to go somewhere else. Our procedure is for the robot to search the graph for representations of places with unexplored exits (in the sense given above), and use the retrace procedures given above to seek out the one which is closest (again, in the sense of the shortest known path from the current location having the fewest links).

5.4.3 Deciding what to do once we get there

In either mode of operation, the robot does eventually come to choose to make a traverse of a certain direction and distance (either the standard exploration step distance, or the distance associated with a particular link in the graph). It then adjusts its heading in the indicated direction, and attempts to move for the given distance, while employing the obstacle-avoidance methods of [21] (which may divert it from the course originally outlined).

Eventually, the scheduled traverse is complete; at that point, the robot must collect odometric and visual data, and determine where it has arrived — i.e., to select an existing graph node which represents its current location, or to create a new one if no existing node can be found.

It is important to note that while we *expect* to find an existing node in case of a retrace (though, as noted above, perhaps not exactly the one we aimed for initially), and while we generally expect to create new nodes in case of exploration, either outcome is possible in either case. While exploring, we may encounter a place which we have entered before, but not from the same direction (the circle case which has already been discussed at some length); while retracing, we may determine that a formerly recognizable place has changed enough that it has become unrecognizable (e.g., someone moved the furniture), or worse, the robot may have gotten lost.

Consequently, the same procedure is used in either case for determining where the robot has arrived (and whether a new node in the graph should be created to represent that place), and in either case, the robot begins its post-arrival processing by invoking that procedure.

5.4.4 Have we been here before?

The robot begins, of course, by making visual observations of its surroundings, and by using the odometric data to estimate its displacement from the center of the place representation from which it started. With this information, broadly speaking, there are two ways in which the robot can verify that a particular place representation in the graph matches its *current* location — either with visual feedback (if the rosette associated with the place representation allows it to get a reliable bearing), or on the basis of pure odometry (which is unfortunately necessary in cases where it cannot — e.g., in case of corridors, as described above).

In either case, one of the more important factors in determining how suitable a given graph node is as a representation of the robot's current location, is to have some estimate of its displacement from the robot's current position. We could, of course, form such an estimate by computing, for each node, the shortest path to the starting location, adding the displacements associated with all the links, and adding the robot's odometric displacement from its starting location to that. (In practice, of course, this computation may be performed by a breadth-first search which takes time linear in the number of graph nodes).

However, as described in section 5.3.2, the results of such a computation are not trustworthy if the paths involved require the traversal of large numbers of graph links, even if the net distances that result seem small. So, once again, we must confront the two paradoxes brought up in that section:

- Odometric data cannot be used in general to recognize that the robot has arrived at a given place, but there are places which cannot be recognized any other way, since the visual environment there does not allow the robot to obtain a position fix relative to visual features.

- Odometric data must be used to recognize certain situations which are guaranteed to come up in sufficiently complex environments (e.g., loops with a long total distance around the loop), even if the places do support a position fix (as visual data will not distinguish two places which look alike, which may be necessary to determine whether the robot has completed a large loop, and if so, which).

No cleverness will take us off the horns of these dilemnas completely, but we can take a guarded approach to the use of odometric data which attempts to minimize the damage they may do.

Specifically, as described in section 5.3.2, it makes sense to trust the results of odometry *provided that the number of graph links traversed — and hence, the accumulated odometric error — is kept within bounds.* We use this approach to minimize the effects of the first paradox — using odometric data to "recognize" places at which we cannot obtain a true position fix, but only if the accumulated error that went into the odometric displacement estimate is within bounds.

However, the second paradox forces us to use odometric data with potentially unbounded error (in practice, bounded only by the length of the longest complete circuit which occurs in the robot's environment). However, the use of such data in connection with place representations which cannot sustain a visual position fix is obviously risking disaster. Accordingly, we use such estimates only to filter and rank matches among place representations which *can* sustain a position fix.

One consequence of adopting this rule, as noted in section 5.3.2 is that the robot may wind up exploring the same corridor, and synthesizing multiple representations for it, more than once, before it finally reaches a place at which it can obtain a position fix. At that point, some amount of repair work may be needed in the graph to establish relationships between the two sets of representations of the corridor (once again, as outlined in section 5.3.2).

Lastly, it is possible that multiple representations may be candidates to represent the robot's destination, and it must choose between them; a procedure is required for doing so.

We now describe these procedures in detail.

Recognizing feature-poor places

We begin with the case of a candidate feature-poor place representation — that is, one whose associated visual environment (i.e., rosette) cannot sustain a full position fix. Such place representations must be verified based only on odometric information, if at all. However, as indicated above, odometric data can only be trusted without a visual check if the path over which it is summed has a relatively small amount of accumulated error — which is to say, that it covers relatively few links in the graph.

Accordingly, the assessment of a candidate feature-poor represntation is based on two computed quantities:

- The (odometrically) estimated displacement from the center of the area represented by the given place representation, to the robot's actual current location.

- The number of graph links traversed in computing the above estimate.

Both of these numbers must be below threshold for the place representation to be a candidate representation for the robot's destination.

The choice of these thresholds is, of course, not arbitrary. In order to keep the graph sufficiently dense as a representation of the environment, we actually want the robot to create a new representation if it is sufficiently distant from the centers of the old ones, even if it has relatively sound estimates of what those distances are. As discussed above (section 5.3.2), the standard exploration step distance is an estimate of how far we would like place representations to be apart ideally; in practice, it is worth allowing for a little sloppiness in either direction, so we fail to accept a candidate place representation if the robot's estimated displacement off center (odometrically in the case under discussion here; visually verified if possible in the case discussed below) exceeds half this distance.

As for the total number of graph links whose traversal we allow, we make a simple conservative choice — this method is not sophisticated, but in practice, does seem to work; see also the comments in the next section on the applicability of Bayesian reasoning to these situations.

Recognizing visually recognizable places

We now consider the case of a place representation which is feature-rich in the sense of the preceding section — that is, it can sustain a position fix. Thus, we can obtain a visual estimate of the robot's displacement from the center of the place representation, which will be quite accurate *if we are actually near the place represented.* However, even if the obstacles currently visible to the robot are a very good match for the configuration stored in the candidate place representation's rosette, we cannot trust the resulting estimated displacement entirely — there may, after all, be two places that simply look alike in the robot's environment, with odometric data being, once again, the only way available to prevent one from being mistakenly "recognized" as the other the first time it is seen (as in figure 5-3). This is, once again, the "circle paradox" discussed in section 5.3.2.

Now, candidate place representations which can sustain a position fix, but do not in fact allow for a high-quality visual match to the local environment, can obviously be rejected without further consideration. Accordingly, a candidate feature-rich place representation is once again evaluated on the basis of two criteria:

- The visually estimated displacement from the center of the area represented by the given place representation, to the robot's actual current location.

- The odometrically displacement of the same quantity.

Note that these two displacements may not be similar — indeed, if the robot has in fact gone in a large loop, their difference will be a fairly good estimate of the accumulated error in the robot's odometry over the loop.

The main use of the odometrically estimated displacement is simply to deal with the case shown by figure 5-3; that is, the case of falsely recognizing one of two places which look alike (as far as the robot can tell) as *being* another.

It is tempting to consider the use of a more sophsiticated Bayesian test, which would weigh probabilities of errors from the odometric and mapping processes, and come up with a prior probability for each place representation, and then choose the one with the larger prior probability of giving the robot's current observations. The problem with such a procedure is that standard MAP methods, which assume Gaussian error distribution for all significant variables, effectively assume that visual error (or failure to recognize some subset of features in a rosette) can be effectively "traded off" against odometric error, and this is not correct. As shown in figure 5-3, if the estimated odometric displacement is sufficiently high, it is possible for the visual data to give a prior probability of 1.0 (i.e., certainty of a match), when the robot is actually nowhere near the place in question. Simply thresholding the odometrically estimated displacement is a crude procedure, but it correctly accounts for this case.

Choosing among multiple candidate representations

One possibility which has not so far been discussed is the possibility that there are *multiple* candidate place representations which meet the above criteria, and the robot must choose between them. In this case, we must choose among the representations.

Our current, rough-and-ready rule for doing so is to choose the one with the smallest estimated displacement off center, unless the other choice is a far superior visual match. This is, at last, how we distinguish between multiple candidate closes for long loops; assuming the two are equally good matches for the robot's current visual environment, the robot will choose the one which is the best fit to the odometric data in its graph.

However, it is worth noting that even if there aren't multiple candidates, the actual estimated displacement will have a substantial amount of odometry noise summed in — and perhaps environmentally reproducible "error" as well (see chapter 2). Thus, it can well be the case that there are loops in the graph resulting from such a match around which the total of the estimated displacements is non-zero.

It is important to realize that this does not constitute an "inconsistency", and no particular measures are needed to "resolve" it, as the robot is not attempting to fix obstacles within a global coordinate system of any kind, and is not actually attempting to represent the global geometry of its environment — it merely maintains a series of overlapping representations of local geometries, which collectively cover the environment, and relies on *those* to be sufficiently close to correct for its purposes.

If all else fails ...

Lastly, we consider what the robot must do if it fails to identify any representation for the place at which it has arrived. This may arise in two cases. Either the robot was exploring, and headed to unknown territory, or it was attempting to retrace its path to a goal location, and arrived at a visually recognizable place where the local obstacles were visually different from what it expected to find.

Either way, in this case, it simply establishes a new place representation, links it to the starting place (with the link carrying the displacement estimated from the odometric data for the robot's traverse). The robot must then resume exploration mode at least temporarily (if it wasn't exploring already) to determine what other links exist to this particular place.

Note that this effectively constitutes a repair mechanism for the graph, in case the environment has changed in some recognizable fashion. An illustration for the way this can arise is as follows:

In the sequence shown in figure 5-6, the robot is retracing its steps through a

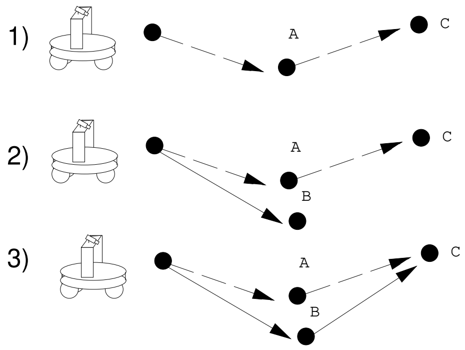


Figure 5-6: Graph repair at work — the robot is seeking place A, which is visually recognizable, but when it gets where that place ought to be, it fails to recognize it; perhaps the furniture has moved. It creates a new representation, B, for the new configuration, and then proceeds to point C, which can be recognized as unchanged. The net result is that a representation for the old configuration of obstacles, A, has been replaced with a more current representation, B.

section of graph where it is currently seeking a recognizable place, A. However, when it arrives there (or rather, when it arrives at a location in the environment with a very small estimated displacement from that place representation's center), it finds a configuration of obstacles which it cannot recognize.

What it does in this case is the same as what it does when no representation with an estimated small displacement exists at all — synthesize a new place representation, and plug it into the graph, giving it a single link to the place from where it started the traverse, then explore to see where that leads. Where it leads in this case is to a place representation which it *can* recognize, indicated by C. When the corresponding link is added to the graph, the net effect is that the old place representation, A, has been superseded by a newer one, B, representing the new configuration of obstacles at the same location.

Creating links

Having identified a representation for the place it has arrived at (or created one), the robot may then create a set of links to augment the graph with information gained through the traverse.

To begin with, if there is no link already between the representations of the robot's initial and final positions, then it must of course create such a link, associating with it the displacement which it estimated odometrically from the actual traverse.

However, as discussed in section 5.3.2, there are cases where it may be useful to form additional links. The operative case is along the lines of figure 5-4; the robot has built two sets of graph nodes, which are duplicate representations of the same physical corridor, and to avoid pathological behavior, must augment the graph with links representing the physical relationships of the places represented by the two sets.

The procedure used to do this is simple: after recognizing a place visually, the robot computes a new set of estimated displacements in the graph resulting from adding the relevant links. If, in the revised graph, two places which were not formerly linked:

- Have a total displacement which is smaller than the standard exploration step distance (so the one may be reached from the other without travelling a long distance and forgoing an alignment check).
- Have that computed that estimated displacement from a number of links which is small enough that the estimate is considered to be reliable, in the sense of section 5.4.4.

then a link is added between those two nodes; this implements the formation of the additional links shown in figure 5-4c.

(It is worth noting that of all the aspects of the robot's behavior yet discussed, this sort of iterative procedure is the least suited to *direct* neural representation. However, that is not to say that there is no functionality in the brain with a similar purpose; indeed, there are hints that Hebb-rule like mechanisms in robot brains may play something of a similar role in rodents. See section 7.3.2).

5.5 Results

5.5.1 The complete system — a simple traverse

We now present an example of the system's performance in a fiarly challenging environment, shown in figure 5-7. This environment has several different locations which are easy to confuse, and many locations (the corridors) where the robot cannot attain a solid position fix. Also, the open areas at the interesections of the corridors are large enough that all features are not visible from everywhere within the intersection — when it is placed at one corner of one of these intersections, the robot cannot see the features at the opposite corner, as they are over the horizon. This means that the robot can pass through an intersection which it has already visited without being able to visually verify that it has done so, which means that it is at risk for failing to close loops.

This traverse is also notable for showing the robot dealing with an error in its odometry (specifically, in its heading readings) of considerable magnitude. Recall that this was done in simulation; all that actually occured was that a random number got added to the difference between the simulated robot's reported heading and the actual heading variable used to track its "physical" position in the reported environment. Nevertheless, the results of the event were disruptive, and the fact that the system is able to cope are grounds for belief that a physical robot using this system could deal effectively with events of this general type.

Now, it is important to note what the graph displays represent. As indicated elsewhere, we do not commit the robot to any particulare global coordinate system, or assign specific global coordinates to any place representation or obstacle. However, when displaying the graph data structure which the robot builds, we are forced to assign 2-d coordinates to place representations for display purposes, because the only

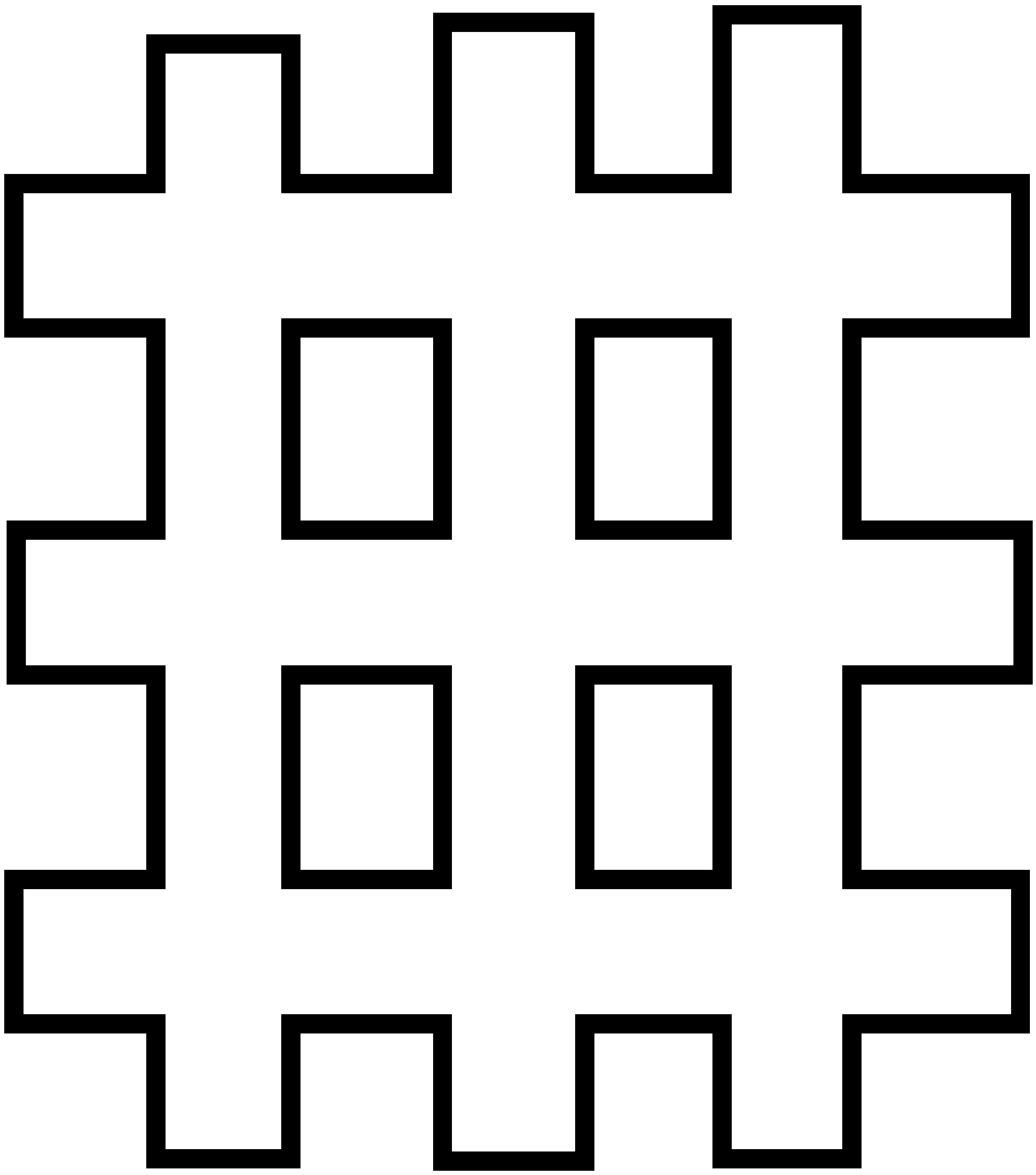


Figure 5-7: A grid-like sample environment.

alternative is to leave them off the page entirely. The procedure for doing so followed in these figures is to compute estimated displacements from some particular place representation (such as the simulated robot's current location), and add suitable offsets to center the resulting structure in the page.

By and large, this gives a picture of the graph which is adequate for allowing an observer to figure out what sort of data structure the robot has built and how that structure is influencing its behavior. However, the procedure does introduce some anomalies. For instance, if there were odometric errors, these will show up in the displays by severely distorting the effective view of certain graph links far from the place representation used as a base for the estimates. Thus, the two final pictures of the graph comprising the robot's map in figure 5-8, figures 5-8i and 5-8j, are actually pictures of the same graph as "seen" from different reference positions. In the former case, the reference position is on the lower left; in the latter, it is on the middle right. In each case, the robot's representation of the local geometry is seen to be correct; however, the coordinates assigned to parts of the environment far from its current have been distorted.

The first anomaly seen in figure 5-8 is in parts c and d of the figure; as indicated above, it can pass through an intersection without recognizing that it has been there before, and has done so, resulting in a doubled representation of the same corridor. A more severe example is shown in parts f and g of the figure — not only does it miss an intersection, but it does so at a time when its heading estimate has accumulated serious drift. The resulting graph has accumulated a whopping amount of global inconsistency by the time the robot manages to (correctly) tie its new representations in with the old ones, in part h, but the final graph (seen in different renderings in parts i and j) still reflects the *local* geometry of the environment correctly, and thus still allows the simulated robot to correctly carry out goal-seeking tasks in the environment.

For further examples, figure 5.5.1 shows a traverse of the playroom-like environment shown in figure 5-10.

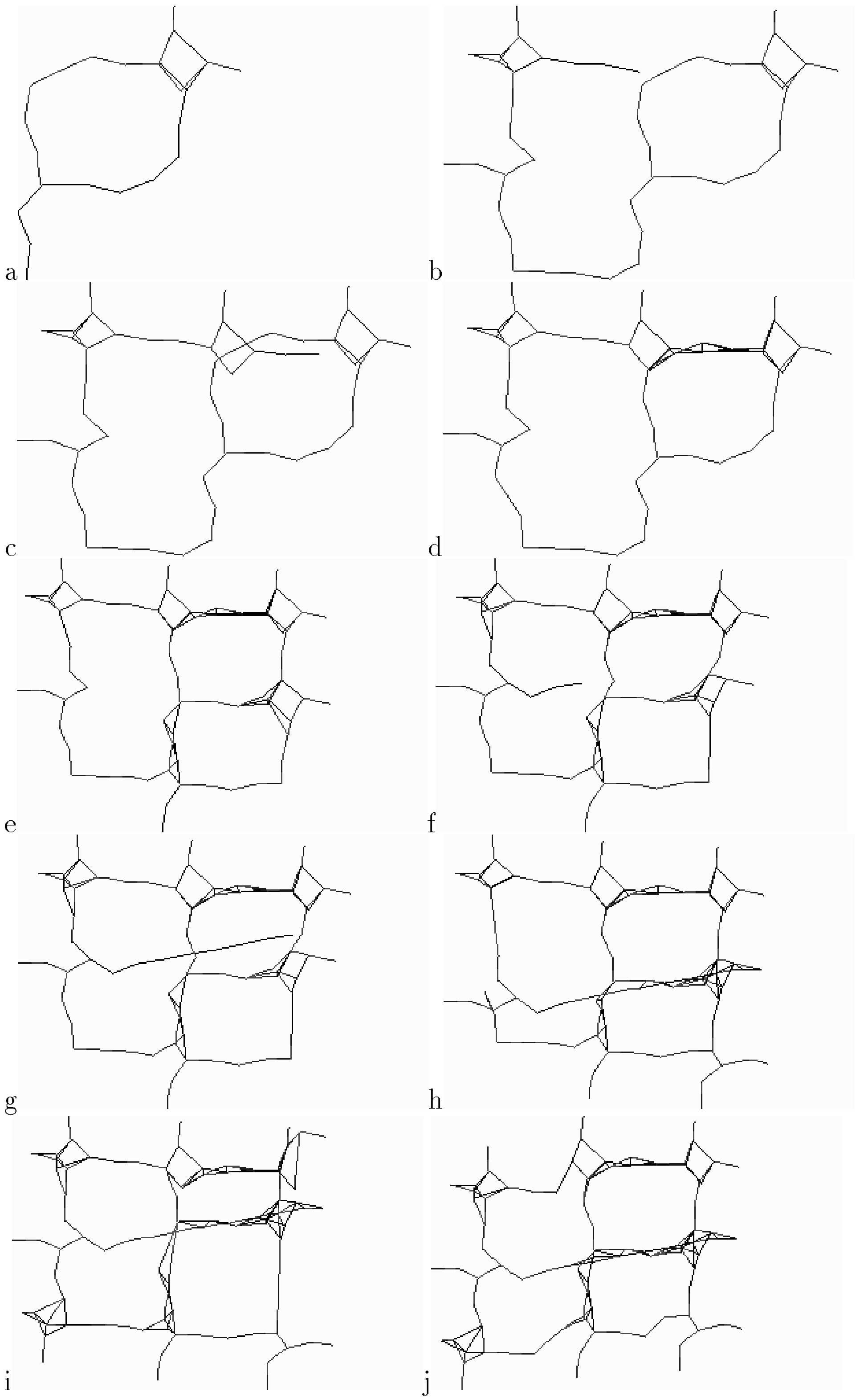


Figure 5-8: Traverse of grid environment.

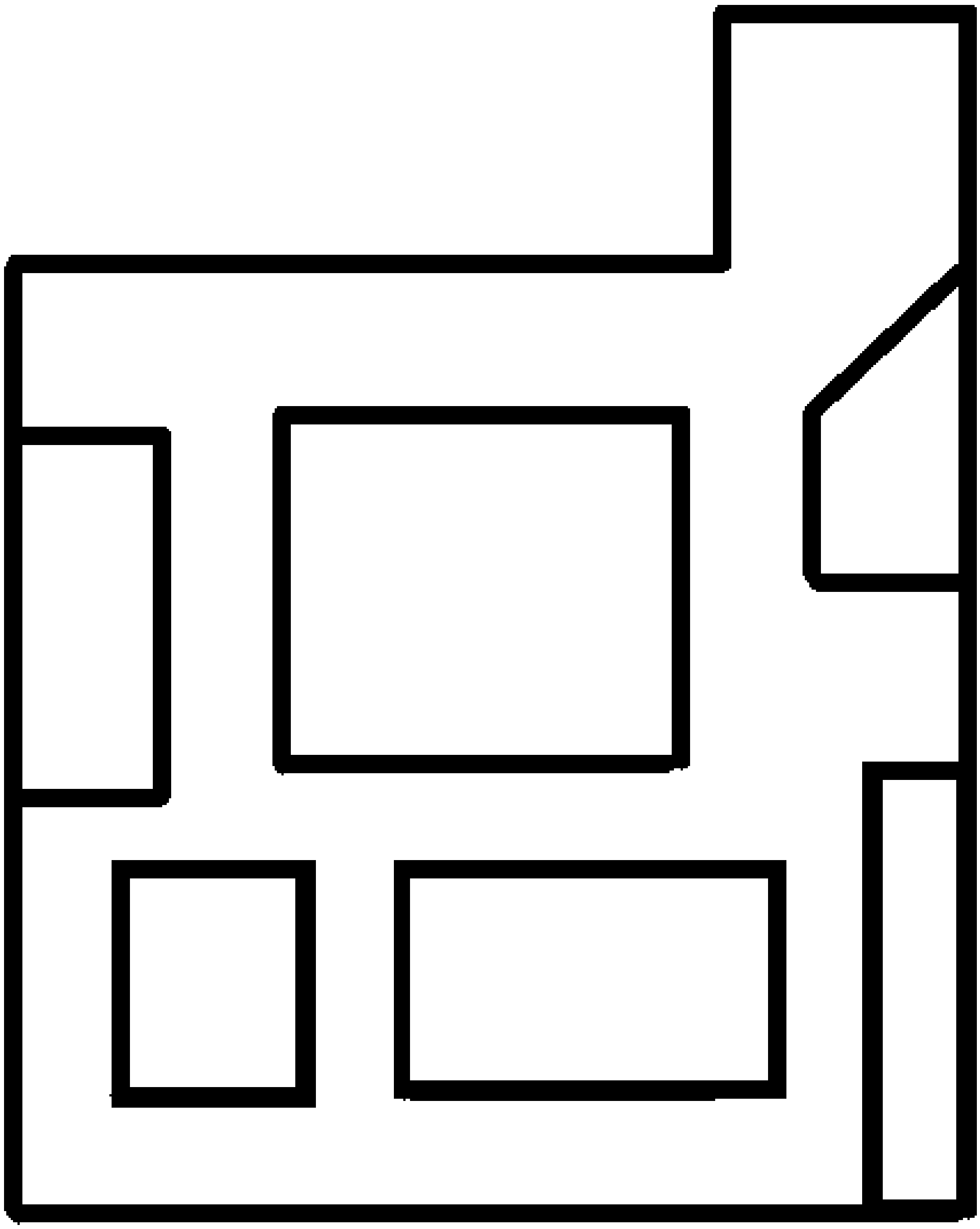


Figure 5-9: A playroom-like sample environment.

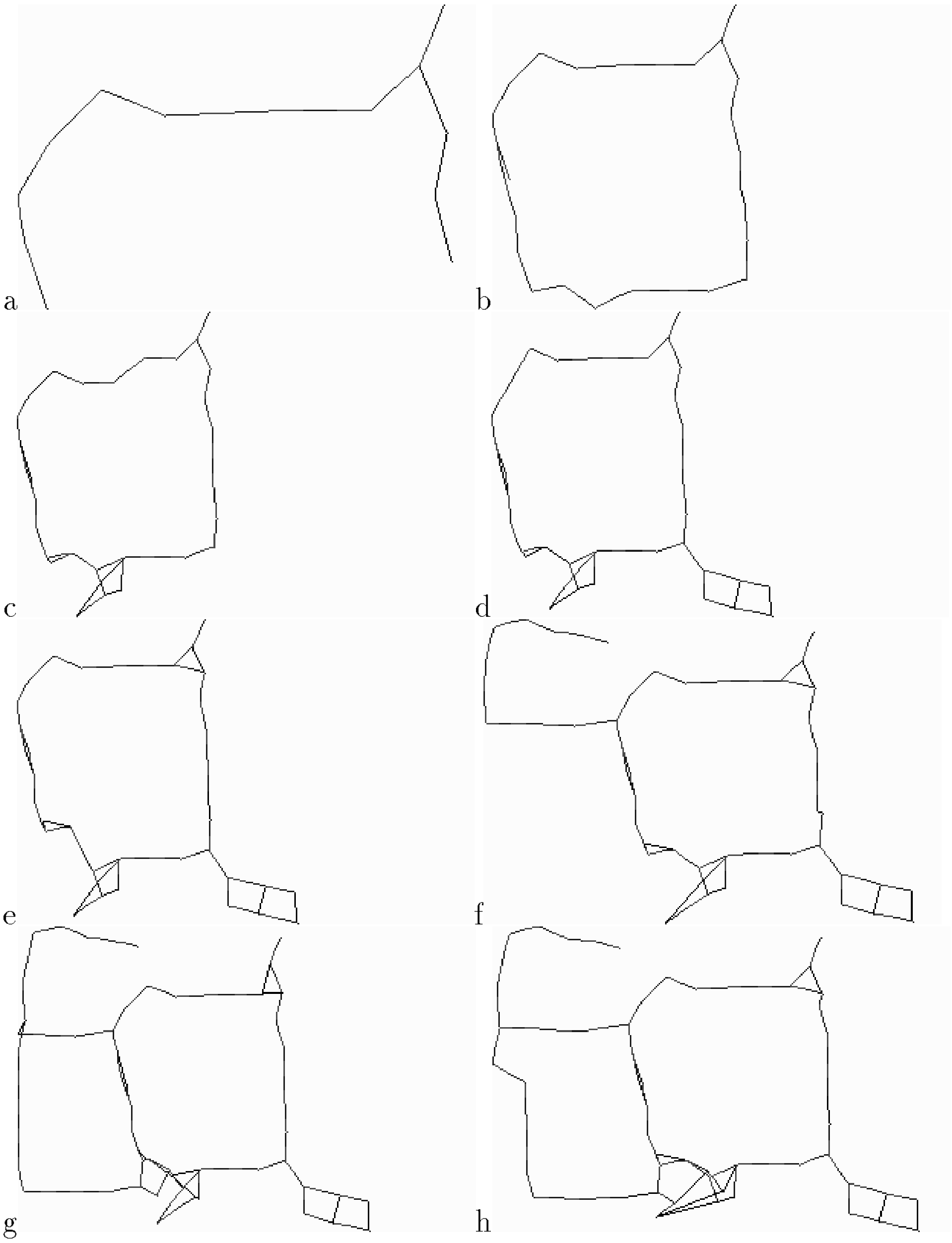


Figure 5-10: Traverse of playroom-like environment.

Chapter 6

Biology Review

6.1 Introduction

The task which our robot attempts to solve — acquiring information which allows it to identify places in a novel environment, and reliably travel between them — is one which needs to be solved by animals as well. It is therefore interesting to compare and contrast what is known of animal behavior, and the machinery which supports it, with the corresponding behaviors and machinery in our robot.

Any such discussion regarding brain functions of mammals necessarily has a highly speculative component. Nevertheless, there is brain activity, particularly in the fore-brains of rodents, which has at least an apparent connection with navigation, and also has interesting parallels to some of the structures and relationships which are built by our robot.

In this chapter, we survey research results concerning the animal brain activity in question. In chapter 7, we go on to consider similarities and differences between the data structures and algorithms actually implemented by our robot, and the computations which *appear* to be taking place in the brain, which we now outline below.

6.2 Place-cell activity in the hippocampus

The most common study subject for neural studies of navigation in mammals, in recent years, has been the rat (and more recently, mice, in which very similar phe-nomenology appear to be present). This is because of the well-known finding that the rat hippocampus (specifically, the CA1 and, to a lesser extent, CA3 gyri) contains "place cells" [36], which respond most strongly when the animal is in a particular place within a familiar environment. The "place fields" of these cells tend to be tightly confined, convex patches of space; cells tend to fire vigorously whenever the animal is in the place field, and much less so when it wanders outside. What makes this activity particularly compelling is that when the animal is roaming freely in a familiar environment, a very substantial fraction (sometimes, more than half) of excitatory, complex-spike units within appear to demonstrate some form of reliable place-field activity within that environment.

We begin this discussion of place field activity by very briefly reviewing experimen-tal findings relevant to place fields themselves. We then discuss evidence regarding non-spatial tasks, and briefly speculate on interpretations of the data which might offer a single consistent explanation for both types of data.

We begin by considering the place-field activity itself.

6.2.1 Place-field basics

The basic finding of the place cell research is that when animals are trained to perform tasks which involve traversing an environment, or even allowed to range freely through one, then cells are found in the hippocampi of the rats (specifically, in the CA3 and CA1 gyri) with properties which appear to be specific to the layout of the environment — that is, they are active only when the animal is in a specific location.

For instance, in the early work described by O'Keefe and Nadel [36], rats were allowed to range freely over an environment with various cues located on the borders to provide landmarks which might allow the animal to distinguish one place from another. So, for instance, an environment might be a space surrounded by a red disk to the north, a buzzer to the east, and a green square to the southwest.

In these circumstances, when recordings are made from cells in the CA1 gyrus of the hippocampi of the free-ranging animals, the preponderance of cells which respond differentially at all appear to fire most vigorously when the animal is in a particular place. These place fields have several properties which have been consistently observed, and which are worth noting here explicitly. They tend to delimit small, compact regions of space — the regions the animal occupies when a cell's activity is above threshold tend to be compact circular blobs, not elongated regions nor star shapes. (In particular, they do not show some of the irregularities one might expect of an aspect-graph based representation). The same cell can have multiple place fields in different environments, if the animal has been trained on both, but in a given environment, it will not be active except when the animal is occupying the unit's place field.

As to the relationships between cells having place-fields, two facts emerge fairly directly. One is that it is generally quite easy to find cells whose place fields overlap — they are *not* forming a tiling of the space the animal has occupied. A more interesting point has to do with topography — while it would be nice to find a topographic representation of space within the (apparently) topographically specific responses can be situated, no one has ever been able to identify one; the relationship between the physical locus of a unit, and its place field (if, indeed, there is any causal relationship at all) is still a mystery.

6.2.2 Degrees of relation to sensory responses

The simplest interpretation of place cell responses is that they have nothing to do with location *per se*, but are simply recording the constellation of sensory stimuli which can be viewed at a given location — information which can be useful in building a map (particularly one of the aspect-graph type), but has no necessary relationship to mapping per se.

The easiest way to tell the difference is to put the animal in the same place but remove or alter the sensory cues. For instance, one approach, followed by Muller and Kubie [34] is to add a barrier to the environment which blocks the view from some particular place to a visible landmark, cells with place fields there will not be affected unless the barrier actually falls within the place field itself. Furthermore, a barrier which *does* cross a unit's place field will abolish the response of the unit even if the barrier itself is transparent plexiglass and all landmarks remain visible.

An particuarly elegant approach to this is found in the work of Best and Thompson [22], who had animals freely explore an environment with visual landmarks, and then turned off the lights. The result was that as the animal continues to wander in the dark (an activity consistent with the rat's natural habits), the activity of cells with place fields is essentially unmodified — the cells fire at an elevated rate whenever the animal is in the physical locations consistent with its place field as determined in the light. Therefore, the responses of the cells are *not* being determined solely by response to a learned combination of certain sensory cues that are immediately present.

What is more important, the continuation of correct place-field responses in the absence of sensory cues demonstrates that the cells exhibiting these responses either are themselves, or are driven by, some sort of a genuine mapping system — that is, one which stores enough information about not only the animal's current location, but its relationships to other locations which the animal has occupied in the past and might travel to again to support a useful dead reckoning capability.

6.2.3 Explicit representation of routing information

The experiments described previously seem to establish that the place-field activity is tied to some notion of "place" which is distinguishable from any property of the animal's direct afferent input itself — it really does have to do with memory for specific places. However, that is not, in and of itself, enough to make up a useful map — one also needs some sort of information about relative connections between the places, which would ideally (at least for the physiologists) be directly manifest as routing-dependant neural activity.

Such activity might take several forms. Indeed, several forms have been found.

Directional place-field responses

Interestingly, when the animal's travels in a particular area tend to be confined to a small set of directions — most typically two, as when the place is a narrow corridor, and the animal can only run along the corridor's axis in one direction or the other — the place field activity is most typically directional as well [32]. That is, a given unit will be active when the animal is at a given place *and* moving in a given direction. However, if the animal is allowed to freely explore an open field, then directional behavior is typically seen only briefly, as the animal learns the environment, and tends to give way to the more frequently discussed purely location-based pattern of behavior, in which a cell's response is more or less the same whenever the animal is within the cell's place field, with little variation in response with changes in the animal's direction of motion.

Routing information in relative timing

So far, the results we have been discussing have all concerned the aggregate responses of place-field units over time. However, neural activity in the hippocampus has a significant timing-related component — specifically, wide-field recordings from the structure in waking animals consistently show a rhythm (called the θ-rhythm) at about 10 Hz. This suggests that the timing of the action potentials of units relative to both each other and the θ-rhythm as a whole may carry information. Recent work shows that this may well be the case.

Specifically, the work of Skaggs et al. [46] shows the following pattern: when an animal is leaving the region of space covered by a unit's place field, that unit will of course begin to fire less often. However, when it does fire, it turns out that those spikes almost always occur in the early phase of the θ-rhythm. Likewise, when the animal is entering the region of space corresponding to a unit's place-field, the unit's action potentials will tend to occur late in the θ cycle. The pattern holds as well when the animal is right in the middle of a unit's place field — that unit will be more active, but its action potentials do tend to occur mostly in the middle of the θ cycle.

In this manner, the activity during a single θ cycle in fact represents not only the *location* which the animal currently occupies, but the *direction* in which it is travelling — with the units representing the origin of the current segment firing first, and those representing the destination firing last.

(As a sidelight, similar pattern may be observed in sleeping animals. In sleep, the hippocampus' long-range potential is still dominated by rhythmic activity, though that rhythm is much faster than in the waking state. If multiple place cells are recorded simultaneously, their relative firing rates, when they are active, will tend to represent physically feasible routes, even though with the animal sleeping they are obviously not correlated directly to its current location or changes thereof).

It is interesting to note that the relatively tight timing of the activity seen here (involving time differences on the order of 30 msec. or so) do not allow for very much time for place-related activity to leave the hippocampus, travel to some other part of the brain (perhaps the parietal lobe), and return. It still remains possible that the activity is driven by the parietal lobe indirectly, but the machinery to support that hypothesis (perhaps some structure of phase-locked loops) would have to be rather complex.

6.2.4 The hippocampus as a spatial organ?

Let us now consider what all this might mean for the function of the hippocampus as a whole. The most straightforward explanation for this sort of activity is that the hippocampus is specialized, at least in the hippocampus, as a mapping organ; indeed, that is the explanation which was posited by O'Keefe and Nadel. However, the evidence available is too scant to make this conclusive. Suppose, for example (as one might, given the consequences of damage to homologous structures in higher animals) that the hippocampus is an organ with a broad role in the formation of new declarative memories. Then, when the animal is trained on navigation tasks (as in most place field experiments), one would certainly expect to see activity related to navigation tasks (just as one would expect to see task-activity whatever the animal was being trained to do).

Furthermore, there are aspects of "place cell" activity which are not directly consistent with the mapping hypothesis. In particular, there is no evident topographic relationship between cells and their place fields. Likewise, when cells have multiple place fields in different pieces of experimental apparatus, the place fields of pairs of cells do not show any consistent relationship.

So, to try to get a complete picture of what is going on in this area, it is useful to look at what is known of activity in this area during tasks with no overt connection to navigation.

6.3 Evidence regarding non-spatial tasks

While responses of the hippocampal gyri containing the place cells have been most extensively studied in spatial tasks, there have also been studies regarding other tasks, which suggest that the hippocampus may be performing some computation with no relation to spatial processing (which, if true, would imply that it is not strictly a spatial-processing organ). Unfortunately, while that evidence is highly suggestive regarding the consequences of hippocampal lesions in primates, it is less suggestive regarding the rats and mice which are of immediate concern to us. However, the evidence is still at least suggestive, and therefore worthy of some study.

6.3.1 Amnesia induced by hippocampal lesion

We begin by considering the well-known consequences of hippocampal lesion in higher mammals, including humans [44]. In monkeys, experiment has shown repeatedly that lesion of the hippocampus brings on severe amnesia, characterized in particular by the failure to learn associations between cues, or sets of cues, and given responses [62, 61].

In humans (in particular, in the famous case of the patient HM, whose hippocampi were lesioned bilaterally in an attempt to control epilepsy, an operation which has never been performed again due to the consequences), these phenomena are seen, but they are seen as aspects of a more general deficit — namely, a dramatic failure of what has come to be called declarative or episodic memory. HM is capable of carrying on conversations normally, so long as his interlocutor remains in the room. However, if they leave the room even briefly, he loses all memory of having ever seen them — an amnesia which is instant, vast and permanent. He has no memory of people who have been caring for him for years; he is, for all intents and purposes, living in the year of his operation, more than thirty years ago.

Similar deficits may be occuring in the monkeys as well; however, declarative memory is difficult to test directly in the absence of language, so the question is hard to assess. There is anecdotal evidence that the animals are socially impaired, which would be a natural sequel of such a deficit.

These deficits, together with similar case reports of humans with naturally occuring lesions including the hippocampus (though not, of course, with surgical precision), have resulted in a widespread view of the hippocampus as a structure specifically associated with the formation of declarative memory, whether the memories in question are of a particularly spatial nature, or not.

One problem with such an interpretation, of course, is that a lesion of the hippocampus does not eliminate declarative memories which have already been formed. Interestingly, there are lesion syndromes in which previously known information is lost, but they are typically associated with lesions elsewhere in the cortex, often in an area which is commonly associated with processing the specific type of information — vision or language, to cite common examples — whose memories have been lost. For instance, in some reports of color agnosias, the patient will report not just a failure to perceive color, but an inability to remember what color was ever like. So, on the basis of lesion evidence, at least, it seems to be that fully formed declarative memory actually resides not in the hippocampus (which can be destroyed without disturbing it), but elsewhere.

A common, though still speculative, interpretation of these data is that the role of the hippocampus is to coordinate the "laying down" of memories elsewhere, by some as yet undetermined mechanism.

6.3.2 Conditioning

Classical conditioning with complex cues is a very well studied behavioral paradigm, and there is a considerable, though controversial, literature on responses in these areas of the hippocampus during classical conditioning with complex cues.

The basic finding is that in straightforward classical conditioning tasks, hippocampal cells acquire a predictive, apparently premotor activity regarding the response, in the presence of the conditioned stimulus [5]. There is also evidence that hippocampal cells can develop responses to particular *combinations* of cues in classical conditioning experiments [57]. Indeed, when Foster et al. [18] recorded from conditioning in an audio task, complex spiking neurons (the same class which give rise to place fields when the animal has navigational tasks) were instead found to be sensitive to the history of the tones presented, with statistically significant interactions extending over the course of at least a few separate trials.

The difficulty in interpreting these findings is that lesion of the hippocampus is not disruptive to classical conditioning per se — animals without hippocampi are perfectly capable of performing the task. Now, this does not, by itself, demonstrate that the hippocampal function is irrelevant to the task — it could be the case that the hippocampus does play a role in conditioning in normal animals, but that in lesioned animals, another system is capable of taking the role. However, it does place very strong conditions on what would be required to produce an experimental demonstration of such a role.

(In detail: An analogy might be drawn to the dual role of the superior colliculus and the frontal eye fields in rhesus monkeys. Either of these is capable of driving near-normal saccadic eye movements in the absence of the others, but a combined lesion of both areas abolishes saccades completely. However, in order to demonstrate a similar role for the hippocampus in conditioning, one would need to identify another area or areas with an equivalent role, and to perform the same set of demonstrations — that lesion of the hippocampus, or of the other area, alone did not block conditioning, but that a combined lesion does. No such experiment has yet been done, and any list of possible "partner areas" would, of course, be highly speculative).

Of course, what is of most interest to us here is not whether the hippocampus in rats and mice is involved in conditioning, but whether it is involved in *any* computation which is distinct from spatial modeling per se (whether that computation is relevant to a particular experimental task or not). Since the hippocampal responses do demonstrably change in conditioning tasks which have no spatial component, that can be taken as evidence that there are non-spatial computations in the hippocampus. Unfortunately, this too is weak evidence, because there has not, yet, been any clear demonstration that the changes in hippocampal responses are due to a computation occuring within the hippocampus itself (as opposed to one of the numerous areas — most of the sensory neocortex and much else besides — which supplies input to the hippocampus). Investigation of hippocampal responses within the mutant mice of Tsien et al. [54, 55], which have LTP blocked in the CA1 gyrus of the hippocampus but are otherwise normal, might help to elucidate this question.

6.4 Hippocampal activity — approaches to interpretation

Let's begin by discussing the actual neural activity which is known to occur in the hippocampus. So far, we have discussed two sorts of activity; one related to navigation and navigational tasks, and one related to classical conditioning, in which the cells appear to form responses which relate specifically to the temporal, rather than spatial sequences of stimuli in the experiment. As described above, there is not yet definitive evidence that the physiological responses seen in classical conditioning tasks are attributable to computations performed by the hippocampus itself. If, however, they are, then the hippocampus in these rodents would clearly be implicated in computation which is *not* purely spatial, contradicting the view of O'Keefe and Nadel [36] that it is purely a mapping organ.

In other words, if the responses in conditioning tasks are, in fact, an expression of the function of the hippocampus, then we would have to suppose that the function of the hippocampus is one that can manifest as "place-related" activity when the animal is exploring a static environment, but can also take the form of evolved responses to combinations of cues in non-spatial conditioning tasks. It is interesting to speculate on what such a function might be.

A natural place to begin such speculation is with the notion of a temporal sequence — such as the units reported on by Foster et al. [18]. Even in the trivial case of a single, simple cue, classical conditioning is concerned with having the animal produce a proper response (the conditioned response) when it sees the conditioned stimulus — a simple temporal sequence. Furthermore, the responses appear evolve additional structure when the cue itself becomes temporally sequenced (though, once again, there is a question as to whether those responses evolve within the hippocampus itself)..

Navigation also is concerned with the association of a set series of responses to a temporal series of cues. There are, of course, several key differences between the sequences of cues involved in navigation, and those involved in the classical conditioning paradigm. The simplest of these, of course, is that the cues being sequenced in navigation include the animal's own recent past, and perhaps possible future actions. More generally, the associations that have to be learned in the fully general place-field case have a higher intrinsic dimensionality than the associations among cues in the classical conditioning experiment — the one dimension of time in the conditioning case, versus the two dimensions of the surface which the animal explores (or three, if time is counted as a relevant dimension in the navigational case).

However, the differences between the two cues can perhaps be bridged by postulating an intermediate sort of experiment. Imagine, if you will, a rat on a roller coaster, with visual and other cues nearby allowing it to distinguish particular bends in the tracks. It is moving, and it knows that it's moving, but it has no choice in the direction of its motion. However, if it formed place fields representing locations along the track, then the sequence of those fields would have some resemblance to the sequence of items learned in classical conditioning — a single, set sequence of cues, which always fire off in the same order, with the samed learned cue or response appropriate at the end. The major difference between such a series, and the cues presented in the sequenced classical conditioning experiments is simply that in the latter case, the animal does not move.

What this "rat on a roller coaster" example is meant to demonstrate is that a structure capable of learning associations as general as those required for navigational tasks is also capable of handling the particular associations required in sequenced classical conditioning tasks as a simple special case, so long as there is nothing wired into the system which *limits* its use to sets of cues in which the animal is making autonomous choices. It is therefore interesting to speculate that the hippocampus is, or contains, such a mechanism, one capable of learning and representing series of cues or actions (including, necessarily, links between the elements of the sequences in question).

This hypothesis is at least consistent with the *absence* of certain features which one might expect to see in an area which is truly specialized for navigation exclusively, such as a predictable relationship between at least local features of the topography of the environment, and that of the area itself. It is also consistent with the observation that, across a considerable spectrum of tasks, experimenters find a very large fraction of units in the hippocampus which have apparently developed responses with a high degree of specificity towards the task in question.

(Finally, it is interesting to speculate that a structure which is capable of learning arbitrary graph structures, might be useful in learning graph structures which don't directly represent an aspect of the real world, including, in the limit, the graph structures of language — but such speculations are way beyond the scope of this thesis).

So much for review and analysis of the data concerning the hippocampus. We now turn to consideration of possible relationships between this activity, and the structures it represents (such as they are), and our robot-based mapping system.

Chapter 7

Biological parallels

7.1 Introduction

The previous chapter has briefly reviewed some research findings concerning activity in the mammalian hippocampus, which seems almost certain to be performing computations related to the modeling of space in rats and mice. (It is possible that the hippocampus is involved in other functions as well even in these animals, and that is quite likely the case in higher organisms — particularly considering the available data from human subjects such as patient HM — but there is no definitive data available).

We now discuss some of the obvious similarities to our robot, and ways in which these similarities reflect the underlying nature of the problems faced by both systems. We then proceed to consider manifest differences, the computational resources required to simulate more brain-like systems (and how that might improve performance), and how this all may reflect the differences between the resources available to the brain and the underlying hardware (or, in the case of the brain, "wetware").

Our robot-based system builds a graph of place representations, which have links to representations of neighboring places. The rodent brain, likewise, contains units which are active when the rodent is in a particular place. There is still no definitive evidence of the exact function of these responses. However, they appear not to result from any direct combination of sensory cues, as evidenced by the continuation of place-cell responses in the absence of cues (as in darkness). Indeed, performance in darkness indicates that the neural system contains some representation of the distances between the represented places as well, or it would be impossible for the representation of the animal's current location to stay current in a "lights out" environment. (We have not demonstrated, of course, that such a representation of relative displacements exists *in the hippocampus* — it might well exist in one of the many areas to which the hippocampal formation has reciprocal connections, or be a property of one of those connections. However, for the observed pattern of activity to be possible at all, it must exist someplace).

To illustrate the analogy between place fields and the place representations built by our system, we can imagine a hypothetical "unit" in our robot which would be active whenever the given place representation. Such a unit would then have a "place field" — that is, it would be active when the robot was near that place, and inactive otherwise; see figure 7.1.

In short, the both the rodent brain and our system contain units which appear to

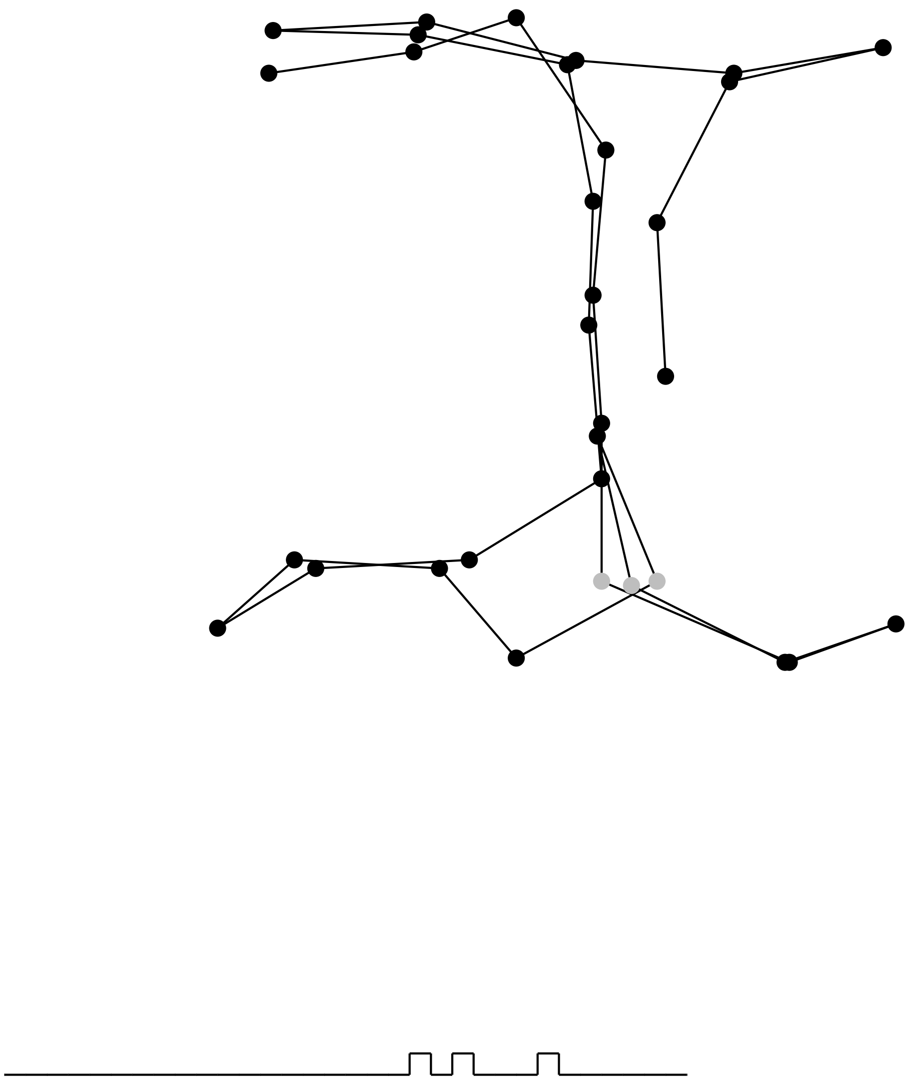


Figure 7-1: “Activity” of a single place representation in the cross environment. Above, we see the position fixes at which the chosen place representation is active, giving some notion of the extent of its “place field”. Below, we see “activity” against time, the closest correspondant in this analogy to a physiological trace of unit activity. There are three instances of activity, corresponding to three separate visits to this physical place during the traverse shown.

represent the presence of the traveler at a particular place, and to represent explicitly their relative displacements in sufficient detail to allow it to tell which place representation, or units, to activate after at least a short stretch of dead-reckoned navigation, with fair accuracy. However, there are some differences, which we now discuss.

7.2 Directionality

One immediately striking difference between our place representations and the place cell activity in the brain is that the latter can be directional. That is, the place-cell units in the brain will often be active only if the animal is in a given place *and moving in a given direction.* In fact, as discussed above, directional activity is highly likely to develop when the directions in which the animal moves are constrained (as in a narrow corridor), and it is the non-directional activity which would most closely correspond to our place representations (associated with no particular direction of transit) which tends to emerge only later and more slowly after the animal has been allowed to explore environments without such constraints.

It is interesting to determine whether this reflects any fundamental distinction between the two models — in particular, can our graph structure sustain directional representations? The answer is that it can, but only with extensive modifications. The reason for this is that the links between place representations in our graph are not themselves directional. This can lead to inappropriate results in searches based on directional nodes — in particular, in retrace mode, the without some notion of the directionality of links, there will be nothing to prevent the robot from attempting to attempting to transit back to a place which will not be recognized because the representation has the wrong polarity.

Nevertheless, such a modification is possible, and with the modification in place, the robot does indeed build a pair of linked linear graphs of place representations, one for each direction, when run down a corridor (as shown in figure 7.2).

To go into more details: the modification requires both place representations themselves and the links between them to be typed as "directional". For place representations, this means that the representation has to be marked with the particular direction in which the robot was facing when the representation was created (i.e., when the rosette was scanned), so that matches can be made to fail when the robot is in the same physical location but oriented differently. (This is a natural restriction for the robot, but for our robot, it is somewhat artificial). Links have to be marked with a designation of which end is the origin, and which the destination — otherwise, the graph-search procedures which direct exploration will try to direct the robot to previously viewed places where it has been, but oriented differently at the time, resulting in serious confusion. (The search procedures themselves have to be updated likewise).

In the simple corridor case, this has no major functional implications; increasing the number of representations by a factor of two will not have a devastating impact on performance. However, in situations where the robot is exploring an open field in which any given place can be approached from a variety of angles, the blowup

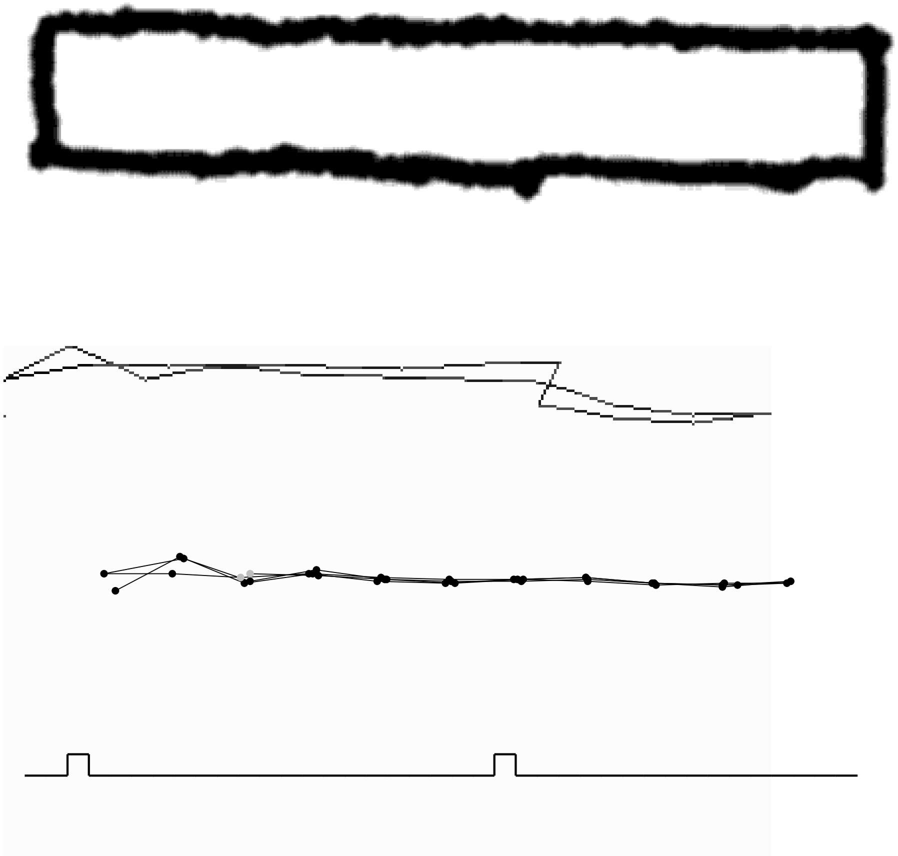


Figure 7-2: A simple corridor environment, and the directional graph built by a modified version of the algorithm, with directional place representations and links. Also shown are place fixes and activity trace of the “unit” corresponding to one directional place representation. Note that the simulated robot traverses the corridor back and forth twice, meaning that it is in each physical locale four times (twice each, out and back). The “unit” is only active twice, showing that it is active selectively depending on direction of transit

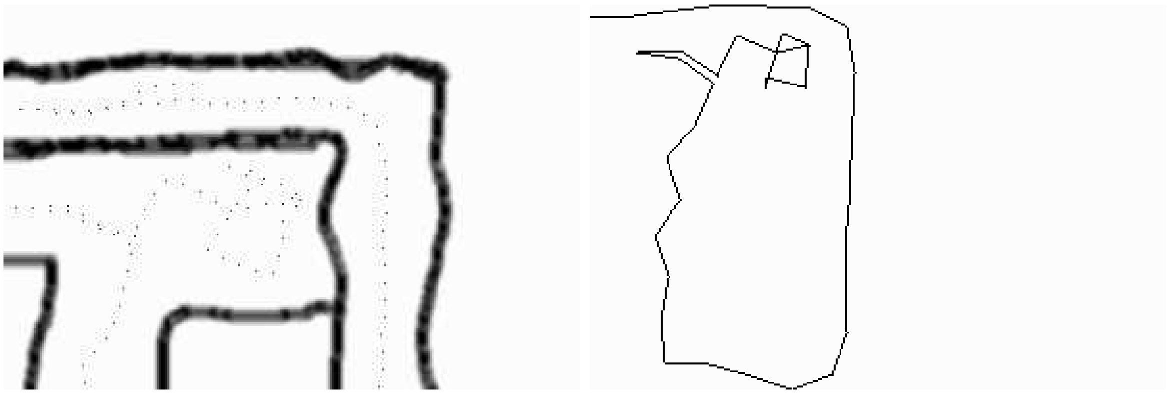
gets more severe. The most direct cause of this is that the number of representations required increases more (with the number of headings from which each place must be approached). However, there is another effect which is even more severe — each of those multiple representations will, when first created, have unexplored exits in the directions of the other, prior approaches, which will yield, at least, an awful lot of redundant exploration to establish links. Both of these effects are on display in the excursion around such a wide area in figure 7.2.

Some of this additional complexity might be eased, of course, by explicitly representing rotation as a separate motor action on the part of the robot, which would correspond internally to transitioning between one of the redundant representations of a given place and another. However, there is no evidence of any such representation in the rodent, and it is interesting to note the reason why. The open-field case in which the directional representations pose the most severe problems for the robot is *also* the case in which rodents acquire hippocampal cells with *non*-directional place fields, in contrast to the apparently more primitive directional place fields seen in simple corridors. It is interesting to speculate that the adoption of nondirectional place fields is a strategy that the brain might adopt for dealing with the same issues of avoiding redundancy and minimizing search which arise when our robot tries to use directional representations in those contexts.

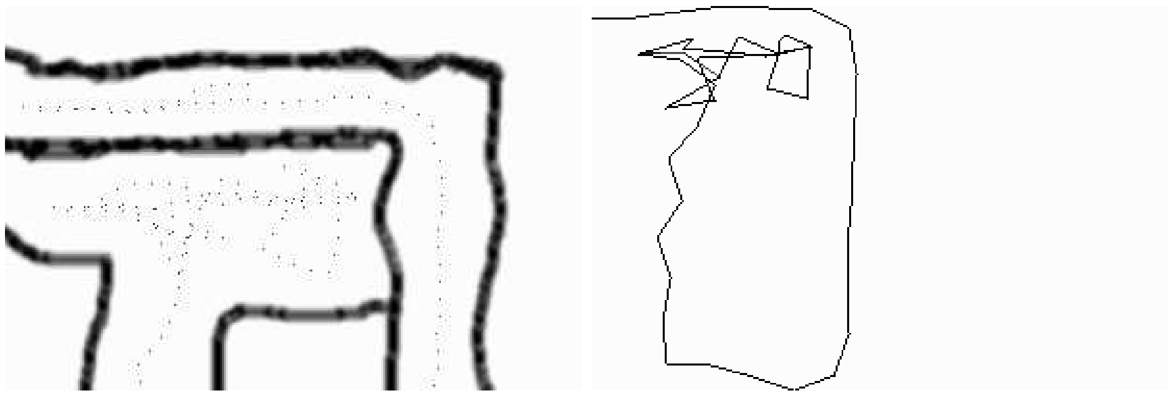
However, even in the simple case where the directional representations seem to work (for both the brain and the robot), they don't seem to confer any immediately obvious advantage. Indeed, in some respects, it seems to confer the opposite — most notably, it multiplies the total number of representations (with the associated increase in total number of nodes, and the probability of blowup due to redundant explorations starting from multiple representations of the same place). We may ask, then, why the brain does things this way.

Two possibilities come to mind. The first is simply a possible limitation on the matching capabilities of the animals. There is experimental evidence that animals do not immediately recognize a place which they have entered before if they are approaching it from an angle which is more than about 90 degrees away from the angle from which the animal had approached the place previously.

If we may momentarily return to the speculations of chapter 6 on the interpretation of the hippocampal activity generally, another explanation suggests itself. Suppose that the hippocampus is, in fact, a general structure for learning associations between situations and the consequences of resulting actions. Then, what may become significant is that the animal is in always in one of two different states in the closed-corridor case (running left *or* running right, but not having to decide which direction to move in). In the open-field case, however, the animal has to choose where it will move from a given place; furthermore, it faces the same choice at any given place no matter from which direction it originally approached that place. So, in the corridor case, the rodent is in two distinct situations with not much in common while at a given location; in the open-field case, it only has one (larger) set of options at any given place. If the representations built by the hippocampus are, in fact, associated with sets of options (and links to possible consequences), this may underlie the difference between the two distinct representations seen in the corridor case, and the single representation seen a)



b)



c)

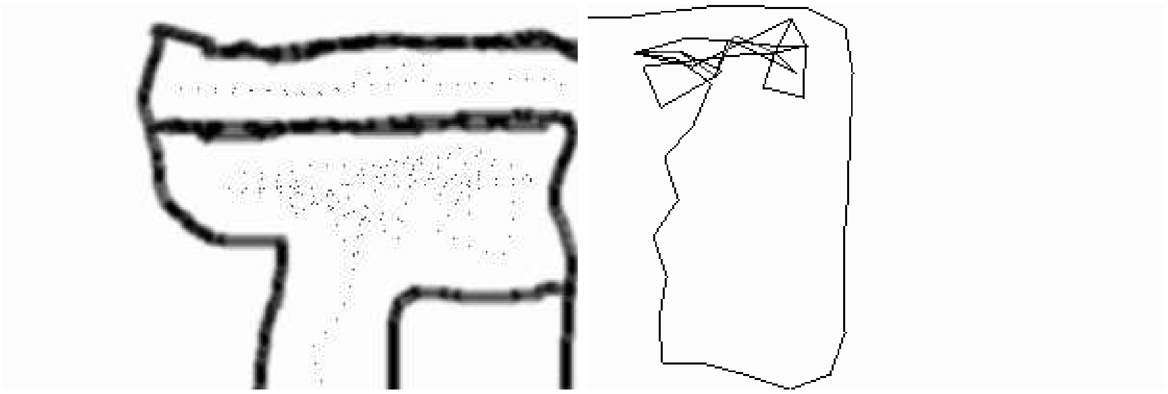


Figure 7-3: Performance of the directional graph algorithm in the case of exploring a large, open area. Note that there are cases, as in the point of map and graph a, where the robot does arrive near enough to a place it has previously encountered, *and facing the same direction*, to recognize it. However, these tend to be very few; the robot quickly acquires a fairly large stock of redundant representations for the same place; exploring all of their unexplored exits would compound the redundancy more before the representation finally stabilized at a massively redundant map.

in the open-field case.

7.3 More brain-like representations

A more systemic difference between the two cases is that our robot algorithms take extensive precautions to avoid building multiple representations of the same place, in order to avoid combinatorial blowup in the total number of representations built (as newly constructed redundant representations have, perforce, unexplored exits which lead to the creation of yet more of them). The brain, by contrast, thoroughly covers the space with an overabundance of representations, and makes no attempt to preserve uniqueness.

This means that the structures residing in the hippocampus of a trained animal (and very possibly elsewhere) are similar in some respects to our system (representing locations and links between them), and different in others (the massive overcommitment). In chapter 5, we have, in effect, extensively discussed the computatational properties of the representations which are built by our robot. In the following sections, we discuss the computational properties a more brain-like version of our representation might possess, and the additional costs and benefits which might be involved in its creation. We then briefly discuss how the differences between the two types of representations, and the associated costs and benefits, might reflect the differences between the underlying computational structures of computers and brains.

First, let us briefly sketch out what a more "brain-like" version of our place graphs might look like. The major difference between the structure itself and the graphs which our actual system builds is, of course, their redundancy. A second immediate difference concerns how the structure itself is used. In our system, the robot represents its own location, in effect, as a single distinguished place representation, and the estimated displacement from its center. The brain is clearly doing something very different — at any given point in time, while the animal is exploring the space it has learned, a very large number of units will be active (some earlier and some later in the theta rhythm in the direction-related pattern discussed earlier). In the graph structure, the most directly analogous sort of representation would be to assign each node in the graph a numerical activity value, and possibly a phase.

The obvious way to get a single estimated location out of such a structure is by averaging the centers of the active representations, weighted by their activations, as discussed earlier. However, that is not necessarily the most useful way to consider such an assignment. It might be more useful to consider the set of activations as a probability distribution over the place representations, in which the activation of each representation indicates a current estimate of the probability that the robot is at (or near) the place represented by that node.

To be processed effectively, such a graph would likewise require a fairly dense set of links. It is interesting to consider the notion of assigning these a probabilistic interpretation as well, in terms of the probability of arriving at a particular place when leaving a particular other place while moving in a particular direction.

Such a "graph with probabilties" would not, of course, directly model the phe-nomenology of the theta-rhythm. We are neglecting it momentarily because the author feels that the distributed nature of the representation, and its massive redundancy, are the differences with the most impact on the computational tractability of the structures themselves, as discussed below.

As discussed below, the actual construction of such a probabilistic model, with full theoretical rigor, would require the development of a fairly elaborate theory, due to the several levels of dependancies that would be involved, and no instances of such a theory are known in the literature. However, even if researchers don't have such a theory immediately, the brain itself might, and it is interesting to consider the advantages that such a theory might confer.

7.3.1 Attractive features of the "brain-like" approach

One immediately interesting feature of the hypothetical probability-web representation of space is that it allows graceful handling of a set of nasty cases which our existing representation cannot handle easily, by spreading the activation among multiple nodes. For example, consider a probability distribution in which there are two local peaks — this would directly represent a case in which the robot could, on the basis of prior data, be in one of two distinct places, which cannot be distinguished on the basis of immediate sensory data.

The usefulness of this possibility becomes apparent when we consider allowing the robot to move around while having such a bimodal notion of its own location. So long as it moved only among spaces which are, once again, indistinguishable (imagine that the robot knows, on some basis, that it is in one of two long corridors, but has no reliable information as to which), the two peaks will be maintained, and will move correspondingly in the graph (as the robot's estimated positions transit up and down the two corridors). However, when the robot reaches a situation where the two peaks correspond to different sensory situations (say, at the ends of the corridors, where the geometry happens finally, to be different), it is easy to determine by direct sensory inspection which of the two representations is incorrect, and the corresponding peak will be suppressed by the failure to fit the sensory data.

In short, the use of probability distributions, rather than a single distinguished position estimate, would allow the robot to maintain and reason about multiple possible hypotheses about its current position. Indeed, there is nothing which limits us to two peaks in the distribution. A limiting case of this is the (actually quite plausible) scenario of the robot starting up knowing that it is in an environment which it has previously occupied, but not knowing where in that environment it has been placed. With the probability-distribution representation, the robot can construct an initial probability distribution based solely on the sensory resemblances of its current location to *each* of the stored representations, and eliminate the ambiguities by exporing, while suppressing peaks which are inconsistent with sensory data acquired at subsequent locations.

(It is interesting to note, in passing, that given a multimodal probability distribution of the sort we are considering, it is absolutely inappropriate to sum the estimates to give a "global position estimate" — that would yield a position midway between the two corridors, which is one place where, given the full probability distribution, the robot most probably is not).

Incidentally, we have just specified a procedure, in the initial stage of recovery from the "unknown location" condition, which is far more expensive than anything which our actual robot does — in particular, we have specified that in the first step of recovery, building the initial distribution, the robot should compute the prior likelihood of each place representation based solely on available sensory data; in effect, an alignment. What makes that procedure expensive for the robot, of course, is that its single microcontroller in the robot can only consider a single place representation at a time.

In our own procedures (in chapter 5), we have tried to avoid matching the available sensory data to anything more than a subset of the available place representations. Furthermore, we are not dealing now with one of our own graphs, with their relatively small number of place representations, but with a multiply redundant version with a higher number of place representations per unit area. This is a very sizable computational problem — beyond what we would expect from the robot itself (though we might consider reducing the cost by using a cruder match, hoping that the redundancy of the representation, leading to mutual support for correct matches, will quickly suppress any mistakes). However, to do several thousand matches is entirely reasonable for the brain, as one or a few neurons can function as a powerful matching engine in their own right. As we will discuss below, this is an instance of a general pattern.

7.3.2 Iterative processes vs. "brain-like" LTP

One immediately striking difference between the two systems is that it is difficult to imagine any direct biological parallel to the link creation procedure of section 5.4.4. Recall that the goal of this procedure is to establish links between two separate chains of representations of the same general physical region. The procedure itself calls for immediate establishment of full-strength connections between designated place nodes which meet a particular, fairly elaborate criterion, which is not something that biological neurons typically do.

However, it is possible that imposing a Hebb-rule like activity upon the network would have the same effect, over the course of several repeated traverses. That is, suppose two separate sets of representations of the same locale arose in a network of the "brain-like" type. By "separate sets", we mean precisely that the two sets of representations are well-connected internally each within each, but that units in one set have no functionally strong connections to units in the other set.

Now, suppose the rodent repeatedly moves through the doubly represented space while Hebb-style reinforcement applies to the (initially weak) connections between the two sets of nodes. Since they both, in fact, represent the same region of the environment, they will tend to fire in coincident fashion. Thus, strong connections between them will be produced by the Hebb-like process in precisely those cases where analogous to the cases where graph links are introduced in our robot by the procedure of section 5.4.4.

Of course, the consequence of somehow disabling the Hebb-rule learning of such a system would be catastrophic — the system might be able to form useful representations of individual places, but it would be unable to form any useful notion of their relationships, since the mechanism for doing so would be blocked. Therefore, even if one could find active place representations inside the system, and they seemed to be precise enough to be useful, it still could not be used for actual navigation.

Interestingly, the actual hippocampi of rodents — and most specifically, the CA1 and CA3 gyri wherein place cells are most often found, have long been known to host a Hebb-like mechanism known as "long-term potentiation", which causes synapses between two neurons to increase in effective strength if the postsynapctic neuron fires shortly after it receives stimulation from the synapse [29].

Further, recent work has been on genetically engineered mice [54, 55, 31], which (thanks to some very clever molecular genetics) lack one of the key proteins for operation of the LTP mechanism — a specific type of membrane-bound glutamate receptor known as the NMDA receptor, on the postsynaptic side – in the CA1 gyrus of the hippocampus. (The brains of the mutant rats being apparently normal elsewhere). Interestingly, examination of these mice shows precisely the pattern above — normal place fields are found for cells in the CA3 field, and unusually large, but not otherwise abnormal, place fields are found in CA1 (perhaps parasitic on the place-field activity of the CA3 inputs), but the mice are simply unable to perform on simple navigation-related tasks, such as the standard "water maze".

7.3.3 Problems with the "brain-like" approach

The discussion above has presumed that the brain is, in effect, implementing some sort of probabilistic model, in which the activations of units can be seen as (representations of) a probability distribution over possible locations (or, e.g., behavioral states) of the rat, and the physiological connections between them made manifest by high cross-correlations in sleep are seen as (representations of) a probability distribution over the possible links between one place in the environment and another.

However, there are deep difficulties in constructing such a theory, particularly if the probabilistic view of links is adopted. The problem comes in trying to manage and update such a structure — simply stating the conditions which the various sets of probabilities would have to meet in order to be consistent with each other becomes quite difficult. Likewise, interpreting the state of such a system, should it fall into a state where all the probabilities are far from their extreme values (close to one or close to zero) might well be an extremely difficult problem. To put it another way, a model in which *everything* is subject to probabilistic interpretation, and *nothing* is known for certain, must be very carefully arranged to avoid a kind of ontological collapse.

However, even with the a sound theory in place, we would still encounter problems with the "brain-like" data structure due to its scale. We have already shown that for one possible application of this data structure, namely, "cold-starting" the robot at an unknown location in a previously known environment. However, the same problem arises in a slightly milder form in the course of ordinary navigation, since, with this structure, many different place representations have supra-threshold activations at any given time, and those at least must be matched against sensory data for their activations to be properly updated. For computers, again, this would pose a severe problem, due to the expense of doing an alignment (although, once again, it might be pointed out that this could be alleviated somewhat by the use of a cheaper, and less reliable, alignment and evaluation procedure). The brain, by contrast, would have no problem doing these operations — or indeed, matching the entire set of active and non-active representations — on a time-scale of tens of milliseconds.

So far, we have just been considering how to use a preexisting "brain-like" map, which has been learned by unspecified means. In particular, forming the graph requires formation of representations of links from one place representation to several (possibly very many) neighbors. Interestingly, this is one case where a purely navigation-related structure might wish to use the structure of the underlying space being explored (in fact, merely the constraint that it *is* a space, with some underlying metric, in which the places close to a given place are close to each other) in order to make a pre-built dense web of local connections. But, if the brain were doing this, we would expect to see that nearby place-field cells have nearby place-fields, and this is not (yet?) seen; no systematic relationship between the physical locations of cells and the locations of their place fields has (yet?) been discovered.

To put it another way — so far as can yet be determined representations of neighboring places, which we might need to link if only to sustain dead-recknoning performance, are not systematically related. Therefore, in order to build the map, the brain (including, possibly, extra-hippocampal structures) must somehow strengthen a link between two hippocampal units which, so far as we can tell, are arbitrarily chosen. It is tempting to merely attribute this to a process implementing something like the Hebb rule for modifying strengths of connections, such as the long-term potentiation which is known to be operating in the hippocampal formation. However, such a process can only operate to strengthen a connection which already physically exists, so this hypothesis would require the prior existence of a very dense web of connections, most of which would be inactive at any given time. The alternatives are that the brain has some way of physically growing new connections as required (by growing new dendrites), or that there *is* in fact some systematic relationship between cells representing neighboring places, which has simply eluded discovery to date.

Another interesting question regarding the maintenance of such a data structure is determining when to tear down old structure or old relationships. The observation that most cells in CA1 tend to acquire some place field in any given environment would seem to imply that there is a need to tear down old structure, lest memories of old environments interfere with learning of, and performance in, the environment currently being learned.

Experimental data seems to suggest that the brain itself may actually have this problem; work in the Wilson lab and elsewhere shows that in cases where the environment itself changes (or, in some cases, where the environment doesn't change but the rat has been disoriented) it appears that the map, as made manifest in the place fields shown by particular cells, can shift radically in fairly short periods of time.

Two questions arise in connection with this phenomenon. The first is the question of determining when to eliminate connections. Here again, the very redundancy of the network may be a saving grace — it may not be bad to be continually paring away at the network and rebuilding it, since the bits which have not been pared away will continue to function, and help to retrain the newly untrained parts; conversely, if the animal does enter a new or changed environment, the old representations will find little sensory support, the newly untrained regions would retrain themselves to suit the new environment, and the new environment would be acquired.

The second, more interesting question, concerns whether it is, in fact, appropriate to tear down known-stable representations. Common sense suggests this is risky; if the animal's environment has been stable for a long period of time, and suddenly changes, it may very well change back, and having to completely relearn a formerly familiar environment would expose the animal to unnecessary danger. Of course, the brain might well deal with this by "laying down" sufficiently stable hippocampal memories elsewhere in cortex, as implied by lesion data, through some as yet unknown mechanism.

7.4 Computers vs. brains?

A theme running through the past few pages has been that the potential advantages of the "brain-like" data structure which we have been envisioning are purchased at the cost of enormous computation — but computation which can proceed in parallel, with communication among units which have (or are acquiring) direct connections, but without substantial global communication. This renders it suitable for the brain, which is made up of units capable of performing fairly complex, but non-iterative, computations on several thousand inputs on a timescale of milliseconds to tens of milliseconds.

This organization contrasts strongly with the organization of modern computers, which is increasingly tending towards a very fast central processor, with a large memory. Interestingly, performance of such computers is increasingly constrained by the inability to get information into and out of the central processor; a key concern in the design of modern supercomputers — those specialized for high-speed evaluation of numerical models — is, in fact, bandwidth to memory. The brain's organization of lots of relatively slow, but highly flexible processors, seems to have advantages in this problem domain.

Chapter 8

Conclusions

The preceding chapters have explored a system for robot navigation, with "short-range" and "long-range" components — the "short-range" components being used by the robot to probe and categorize its immediate surroundings, and the "long-range" components to organize the "short-range" percepts into a useful persistent representation of the robot's entire environment (including those portions of it which are out of immediate view). We have also discussed possible parallels (and some clear differences) between this system and the navigation mechanisms which may occur in the brains of mice and rats (particularly regarding the "place-field" activity within their hippocampi). In this chapter, we take stock of the complete system, and discuss where further elaboration of the results presented here may be useful.

8.1 What have we accomplished?

We begin by examining exactly what we have and have not accomplished, purely within the domain of robotics, and then discuss some possible extensions.

8.1.1 Short-range mapping

To begin with, the short-range mapping technology discussed in this thesis allows a (relatively) cheap camera, together with some very simple computational postprocessing, to be used as a simple and very effective range sensor, within certain constraints which are typically satisfied by common office environments. This method has been tried in real robots, with good results; indeed, odometric noise is the major observed noise source. In some cases (e.g., obstacles off the floor) the system will be subject to small, but reproducible errors — however, these errors are typically less severe than for, say, sonar, which is far more commonly used as a range sensor (and which may miss the presence of obstacles entirely if they are specular reflectors of sound).

However, the performance of the system stays good only so long as the constraints are obeyed, and experience with the obstacle avoidance strategies which our rangefinder extends suggests a few pitfalls. For example, the obstacle avoidance system is often tripped up by spurious obstacles which are detected when sunlight from an open office doorway falls across the carpet of a corridor, and the high contrast between the bright and shadow regions triggers the edge detector [21].

More generally, there is a wide class of transient obstacles in many environments which it would be good to avoid placing into any map, as they are more likely than not to have moved by the time the robot returns to its present location. For example, people may well be present in corridors the robot's environment at the time it goes exploring, but they are probably there to go someplace, and will probably not still be standing around in the corridors when the robot comes back.

A coherent strategy for dealing with transient obstacles would therefore be very useful to the system. Unfortunately, the most essential foundation for any such strategy is finding some way to tell the transient obstacles from the others. The robot control system from Horswill's thesis [21] did have a crude sort of "people detector", based on detecting tall vertical forms which might have been a person's legs, which was used to initiate the "tour pick-up" behavior pattern. However, this person detector was crude and somewhat prone to misfire — when the system was in operation, it was not uncommon for the robot to pause and ask some unsuspecting table or water cooler whether or not it wanted a tour of the lab. A more reliable approach would be desirable, but may not be easy to provide.

8.1.2 Long-range mapping

The long-range mapping algorithm presented here has been tried extensively in simulation, but due to breakdowns in the robot hardware, has never been tried in actual robot hardware. Nevertheless, it has some interesting properties, which should be useful in real applications.

Approach to the use of metric data

To begin with, the general philosophy on the use of metric data — keeping track of local relationships without trying to explicitly reconstruct a single global coordinate frame — is somewhat unusual, and avoids some subtle pitfalls. For example, it can easily be the case that the robot's wheels reliably slip more in some parts of the environment than in others, and where that is the case, it would be more of a surprise if the odometrically estimated distance around a loop in the environment were close to zero than if it weren't. Likewise, calibration errors in the robot's odometry may lead to systematically skewed sensory readings. Note that since the slip on any given surface (or the poor calibration, etc.) is reliable, the local observations themselves may well be quite reproducible — it's just that they're systematically inaccurate.

The system presented here adapts very well to this situation — it uses the reproducible local odometric readings as they occur, and simply isn't bothered by the inconsistencies it would face if it ever tried to determine the positions of any of the obstacles (or for that matter, its own position) in a single, global coordinate frame.

However, many robot mapping systems do attempt to assign the obstacles set coordinates within a 2-d frame, and for them, the "reliable slip" situation creates a serious difficulty. By the nature of its representations, it must assign some 2-d Cartesian coordinates to the locations along the path. The differences between these coordinates are then estimates of the displacements between the locations in question.

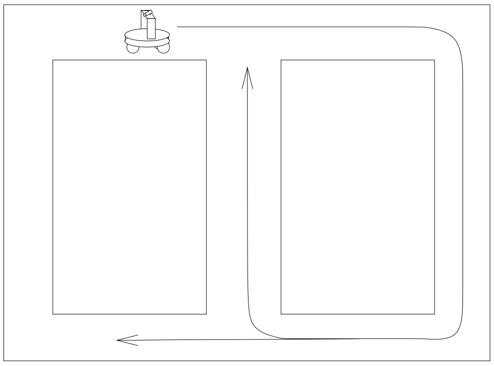


Figure 8-1: The robot has two choices as to which way to proceed. One has the potential to lead back to explored territory; the other does not. Right now, its choice is random; it might be better to have it systematically biased toward the former.

However, note that by the very nature of the representation — that the starting and ending points of a loop, being the same location, must have the same coordinates within the assumed global reference frame — these estimates of the displacements between the locations comprising the loop must sum to zero. Therefore, they cannot accurately reflect the actual observations of those displacements, which reliably do *not* sum to zero. In other words, a system using such a representation must, in these cases, somehow systematically distort reliable observations in order to fit them within the constraints imposed by its own representational framework. By (nearly completely) avoiding the use of a global coordinate frame, our system neatly sidesteps this difficulty.

Further work

However, some problems remain with the system, and there are plenty of opportunities for further work. To begin with, the system should be tried out at some point in an actual robot. However, even in simulation, there are some possible improvements which suggest themselves.

Cues to uniqueness of location To begin with, the rosette data structure, while reliable and reproducible, does not nearly exhaust the amount of data available from vision. Many other mapping systems have used visual methods to yield some sort of a "hash code" for the surroundings, which includes much more information than simply range to nearest obstacle (e.g., contents of the walls). Incorporation of such a strategy might be very helpful, particularly in cases where the rosette itself is inadequate to gain a complete position fix.

Exploration strategy On a more mundane level, the robot's choices of exploration direction, at a place with multiple exits, is, at present, arbitrary. This sometimes results in suboptimal patterns of behavior. For example, consider the case shown in figure 8.1.2. In this instance, the robot has two ways to proceed — however, one offers at least the prospect of a return to previously explored territory (by immediately closing a loop). In this instance, of course, the other choice will also result in a return to previously explored territory, but only by following a longer loop, which will result in greater accumulation of error. In this case, at least, the robot might be better off biasing its choice towards the former alternative.

8.2 Parallels to biology

From purely robotic concerns, we now turn to the subject of chapter 7 — possible parallels between our system and the neural support for navigation in the brains of rodents, as apparently revealed in the place fields of cells in their hippocampi.

8.2.1 A summary of the comparison

The major point of similarity between the two systems (robotic and biological) is that the biological system *appears* to be involved in the construction of a graph-based representation of space, just as our robot does. It is worth noting, though, that the basic nature of the two graph-based representations differs somewhat.

Specifically, the biological graph (if such it be) is clearly characterized by massive overrepresentation of any given point in the robot's environment (or at least, this is the common result within the somewhat confined environments which are commonly used to probe the behaviors of these units within the laboratory). This contrasts rather strongly with the robotic representation, which deliberately tries to avoid creating multiple representations of the same confined area when it can manage to do so. The possible functional implications of this overrepresentation (both the potential problems in dealing with very large redundant graphs, and the potential benefits that might be gained if tractable algorithms could be found to deal with those problems) are discussed in some detail in chapter 7.

However, it is also worth noting that some of the advantages of a graph-based representation (e.g., those noted above in section 8.1.2, such as robustness in the face of uncertain, or even systematically erroneous, odometry) apply equally to either sort of graph.

8.2.2 Further work

One potentially *very* interesting piece of future research would be to actually build a neural network along the lines of the "brain-like" network speculations in chapter 7. Among the possibilities such a system might offer is the prospect of a direct comparison between the behavior patterns of its units, and the actual behavior of hippocampal place cells (as opposed to the somewhat indirect comparisons of chapter 7). Likewise, from the robotic perspective, it would be very interesting to see if the functional benefits of redundant overrepresentation and weak, "probabilistic" links described there could, in fact, be realized by the model system.

8.3 Finally

In short, this thesis describes a system which offers some new and interesting approaches to the problem of having a robot build reliable maps of its environment, and offers interesting parallels between its methods and those which appear to take hold in biological systems. However, neither problem domain is exhausted thereby, nor is that likely to happen for some time to come.

Bibliography

[1] Minoru Asada. Map building for a mobile robot from sensory data. *IEEE Trans. Syst. Man Cybern*, 20(6):1326–36, 1990.

[2] Ivan A. Bachelder and Allen M. Waxman. Movile robot visual mapping and localization: A view-based neurocomputational architecture that emulates hippocampl place learning. *Neural Networks*, 7(7):1083–1100, 1994.

[3] Ivan A. Bachelder and Allen M. Waxman. A view-based neurocomputational system for relational map-making and navigation in visual environments. *Robotics and Autonomous Systems*, 16:267–289, 1995.

[4] Craig Becker, Joaquin Salas, Kentaro Tokusei, and Jean-Claude Latombe. Reliable navigation using landmarks. In *International Conference on Robotics and Autonomous Systems*, pages 401–406. IEEE, IEEE, 1995.

[5] T. W. Berger, P. C. Rinaldi, D. J. Weisz, and R. F. Thompson. Single-unit analysis of different hippocampal cell types during classical conditioning of rabbit nicitating membrane response. *J. Neurophysiol.*, 50:1197–1219, 1983.

[6] Margrit Betke. Learning and vision algorithms for robot navigation. TR 671, MIT Laboratory for Computer Science, 1995.

[7] Margrit Betke and Nicholas C. Makris. Fast object recognition in noisy images using simulated annealing. In *Proc. Fifth Intl. Conf. on Computer Vision*, Cambridge MA, June 1995.

[8] Thomas Breuel. Geometrical aspects of visual object recognition. TR 1374, MIT AI lab, 1992.

[9] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, 1986.

[10] Todd Cass. Feature matching for object localization in the presence of uncertainty. memo 1133, MIT AI lab, 1990.

[11] Raja Chatila and Jean-Paul Laumond. Position referencing and consistent world modeling for mobile robots. In *International Conference on Robotics and Autonomous Systems*, pages 138–145. IEEE, 1985.

[12] Jonathan Connell. A colony architecture for an artificial creature. TR 1151, MIT AI lab, 1989.

[13] Xiaotie Deng and Andranik Mirzaian. Competitive robot mapping with homogeneous markers. *IEEE Trans. Robot. Autom.*, 12(4):532–542, 1996.

[14] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robotic exploration as graph construction. *IEEE Trans. Robot. Autom.*, 7(6):859–864, 1991.

[15] Gregory L. Dudek. Environment representation using multiple abstraction levels. *Proc. IEEE*, 84(11):1684–1704, 1996.

[16] Hugh F. Durrant-Whyte. An autonomous guided vehicle for cargo handling applications. *Int. J. Robot. Res.*, 15(5):407–440, 1996.

[17] Hugh F. Durrant-Whyte, Edward Bell, and Philip Avery. The design of a radar-based navigation system for large outdoor vehicles. In *International Conference on Robotics and Autonomous Systems*, pages 764–69. IEEE, IEEE, 1995.

[18] Thomas C. Foster, Edward P. Christian, Robert E. Hampson, Kenneth A. Campbell, and Sam A. Deadwyler. Sequential dependancies regulate sensory evoked responses of single units in the rat hippocampus. *Brain Res.*, 408:86–96, 1987.

[19] William Eric Leifur Grimson. *Object Recognition by Computer: the Role of Geometric Constraints.* Artificial Intelligence. MIT Press, Cambridge, Mass., 1990.

[20] J. Horn and Gunther Schmidt. Continuous localization of a mobile robot based on 3d-laser-range-data, predicted sensor images, and dead reckoning. *Robotics and Autonomous Systems*, 14(2):99–118, 1994.

[21] Ian Horswill. Analysis of adaptation and environment. *Artificial Intelligence*, 73(1):1–30, 1995.

[22] B. Leonard Jones, B. L. McNaughton, and C. A. Barnes. Long-term studies of place field interrelationships in dentate gyrus neurons. *Soc. Neurosci. Abstr.*, 11:1108, 1985.

[23] Eric Krotov and Regis Hoffman. Terrain mapping for a walking planetary rover. *IEEE Trans. Robot. Autom.*, 10(6):728–739, 1994.

[24] Eric Krotov and Reid Simmons. Perception, planning and control for autonomous walking with the ambler planetary rover. *Int. J. Robotics Res.*, 15(2):155–180, 1996.

[25] Benjamin Kuipers and Todd Levitt. Navigation and mapping in large-scale space. *AI Magazine*, 9(2):155–80, Spring 1988.

[26] Andreas Kurz. Alef; an autonomous vehicle which learns basic skills and constructs maps for navigation. *Robotics and Autonomous Systems*, 14:171–183, 1995.

[27] John J. Leonard and Hugh F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Trans. Robot. Autom.*, 7(3):376–382, 1991.

[28] John J. Leonard, Hugh F. Durrant-Whyte, and Ingemar J. Cox. Dynamic map-building for an autonomous mobile robot. *International Journal of Robotics Research*, 11(4):269–85, 1992.

[29] Daniel V. Madison, Robert C. Malenka, and Roger A. Nicoll. Mechanisms underlying long-term potentiation of synaptic transmission. *Ann. Rev. Neurosci.*, 14:379–398, 1993.

[30] Maja Mataric. A distributed model for mobile robot environment-learning and navigation. TR 1228, MIT AI lab, 1990.

[31] Thomas J. McHugh, Kenneth I. Blum, Joe Z. Tsien, Susumu Tonegawa, and Matthew A. Wilson. Impaired hippocampal representation of space in ca1-specific nmdar1 knockout mice. *Cell*, 87(7):1339–1350, 1996.

[32] B. L. McNaughton, C. A. Barnes, and J. O'Keefe. The contributions of position, direction and velocity to single unit activity in the hippocampus of freely-moving rats. *Exp. Brain Res.*, 52:41–49, 1983.

[33] Hans Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2):61–74, 1988.

[34] Robert U. Muller and John L. Kubie. The effects of changes in the environment on the spatial firing of hippocampal complex-spike cells. *J. Neurosci.*, 7:1951–1968, 1987.

[35] Toshihiro Nishizawa, Akihisa Ohya, and Shin'ichi Yuta. An implementation of on-board position estimation for a mobile robot. In *International Conference on Robotics and Autonomous Systems*, pages 395–400. IEEE, IEEE, 1995.

[36] J. O'Keefe and L. Nadel. *The Hippocampus as a Cognitive Map.* Oxford University Press, 1978.

[37] Sageev Oore. A mobile robot that learns to estimate its position from a stream of sonar measurements. Master's thesis, University of Toronto, 1995.

[38] Giuseppe Oriolo, Marilena Vendittelli, and Giovanni Ulivi. On-line map building and navigation for autonomous mobile robots. In *International Conference on Robotics and Autonomous Systems*, pages 2900–2906. IEEE, IEEE, 1995.

[39] Nageswara Rao and S. S. Iyengar. Autonomous robot navigation in unknown terrains: Incidental learning and environmental exploration. *IEEE Trans. Syst. Man Cybern.*, 20(6):1443–1449, 1990.

[40] R. L. Rivest and R. E. Schapire. A new approach to unsupervised learning in deterministic environments. In *Proceedings of the Fourth International Workshop on Machine Learning*, 1990.

[41] R. Andrew Russell. Laying and sensing odor markings as a strategy for assisting mobile robot navigation tasks. *IEEE Robotics and Automation*, 2(3):3–9, 1995.

[42] Jorge L. C. Sanz, editor. *Image Technology*. Springer Verlag, Berlin, 1996.

[43] Bernt Schiele and James Crowley. A comparison of position estimation techniques using occupancy grids. *Robotics and Autonomous Systems*, 12:163–171, 1994.

[44] W. B. Scoville and B. Milner. *J. Neurol. Neurosurg. Psychiatr.*, 20:11–21, 1957.

[45] Kristian Simsarian, Thomas Olson, and N. Nandhakumar. View-invariant regions and mobile robot self-localization. *IEEE Trans. Robot. Autom.*, 12(5):810–815, 1996.

[46] William E. Skaggs, Bruce L. McNaughton, Matthew A. Wilson, and Carol A. Barnes. Theta phase precession in hippocampal neuronal populations and the compression of temporal sequences. *Hippocampus*, 6:149–172, 1996.

[47] Anthony Stentz and Martial Hebert. A complete navigation system for goal acquisition in unknown environments. *Autonomous Robots*, 2:127–45, 1995.

[48] Andrew Stevens, Michael Stevens, and Hugh F. Durrant-Whyte. Oxnav — reliable autonomous navigation. In *International Conference on Robotics and Autonomous Systems*, pages 2607–2612. IEEE, IEEE, 1995.

[49] Raj Talluri and J. K. Aggarwal. Mobile robot self-localization using model-image feature correspondance. *IEEE Trans. Robot. Autom.*, 12(1):63–77, 1996.

[50] Jun Tani. Model-based learning for mobile robot navigation from the dynamical systems perspective. *IEEE Trans. Syst. Man Cybern.*, 26(3):421–436, 1996.

[51] Kenji Tashiro, Jun Ota, Yeuan C. Lin, and Tamio Arai. Design of the optimal arrangement of artificial landmarks. In *International Conference on Robotics and Autonomous Systems*, pages 407–413. IEEE, IEEE, 1995.

[52] C. J. Taylor and D. J. Kriegman. Vision-based motion planning and exploration algorithms for mobile robots. In *Workshop on the Algorithmic Foundations of Robotics*, 1994.

[53] Mark Torrance. Natural communication with robots. Master's thesis, M.I.T., 1994.

[54] Joe Z. Tsien, Dong Feng Chen, David Gerber, Cindy Tom, Eric H. Mercer, David J. Anderson, Mark Mayford, Eric R. Kandel, and Susumu Tonegawa. Subregion- and cell type-restricted gene knockout in mouse brain. *Cell*, 87(7):1317–1326, 1996.

[55] Joe Z. Tsien, Patricio T. Huerta, and Susumu Tonegawa. The essential role of hippocampal ca1 nmda receptor-dependant synaptic plasticity in spatial memory. *Cell*, 87(7):1327–1338, 1996.

[56] M. Weigl, B. Sremiatkowska, K.A. Sikorski, and A. Borkowski. Grid-based mapping for autonomous mobile robot. *Robotics and Autonomous Systems*, 11(1):13–21, 1993.

[57] Cynthia G. Wible, Robert I. Findling, Matthew Shapiro, Eric J. Lang, Sharon Crane, and David S. Olton. Mnemonic correlates of unit activity in the hippocampus. *Brain Res.*, 399:97–110, 1986.

[58] Yasuhi Yagi, Yoshimitsu Nishizawa, and Masahiko Yashida. Map-based navigation for a mobile robot with omnidirectional image sensor copis. *IEEE. Trans. Robot. Autom.*, 11(5):634–648, 1995.

[59] Erliang Yeh and David J. Kriegman. Toward selecting and recognizing natural landmarks. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1995.

[60] Zhengyou Zhang and Olivier Faugeras. A 3d world model builder with a mobile robot. *Int. J. Robotics Research*, 11(4):269–85, 1992.

[61] Stuart Zola-Morgan, Larry R. Squire, and David G. Amaral. Lesions of the amygdala that spare adjacent cortical regions do not impair memory or exacerbate the impairment following legions of the hippocampal formation. *J. Neurosci.*, 9:1922–1936, 1989.

[62] Stuart Zola-Morgan, Larry R. Squire, and David G. Amaral. Lesions of the hippocampal formation, but not lesions of the fornix or the mammillary nuclei, produce long-lasting memory impairment in monkeys. *J. Neurosci.*, 9:897–912, 1989.