Passive Map Learning and Visual Place Recognition

Sean Philip Engelson

YALEU/CSD/RR #1032
May 1994

Abstract
Passive Map Learning and Visual Place Recognition

Sean Philip Engelson
Yale University
1994

An autonomous mobile robot needs to learn about its environment while pursuing its other goals. This dissertation investigates the problem of *passive* map learning, where the robot is not allowed to rely on exploration for learning the structure of its environment. Forbidding exploration makes the problem more difficult, since the true state of the world cannot always be inferred from the incomplete and noisy information available at any one point in time. If the robot cannot actively explore in order to verify the state of the world, the mapping system will inevitably make errors. Two main ideas contribute to the operation of the passive mapping system described here: (a) robot localization and map updating follows a principle of least commitment, with a proper treatment of estimation ambiguity, and (b) errors in mapping are allowed, but explicitly diagnosed and eventually corrected. The map representation used is path-based, but incorporates metric information to provide the constraints necessary for passive mapping. These metric constraints allow the system to diagnose mapping errors and to correct them. The system has been shown empirically to converge on correct maps of its environment in a realistic simulation.

Another important part of map learning is place recognition, or deciding where the robot is in its map. This dissertation also presents a novel image-based method for using vision to aid in place recognition, using *image signatures*. An image signature is an array of values derived by evaluating a measurement function over large blocks of pixels. Measurements are chosen to be characteristic of a location yet invariant over different viewing conditions. Signature matching is performed quickly by element-wise comparison. Experiments on a large image corpus demonstrate that image signatures give an accurate and efficient method for visual place recognition.

Passive Map Learning
and
Visual Place Recognition

A Dissertation
Presented to the Faculty of the Graduate School
of
Yale University
in Candidacy for the Degree of
Doctor of Philosophy

by
Sean Philip Engelson
May 1994

To my parents

Acknowledgements

First and foremost, I want to thank my advisor, Drew McDermott, who taught me, in more ways than I can express, what research in AI is all about. His precise intuition helped clarify my often murky ideas, so that they could be properly verified (or discarded, as the case may be). His constant insistence on clarity and simplicity also greatly improved my writing.

Greg Hager has been a great teacher, colleague, and friend. He helped me learn to keep my head out of the clouds and focus on the bottom line. He also taught me a great deal about robotics and vision. Dave Kriegman's reading of this dissertation was also most helpful. I'd also like to thank my outside reader, Lynn Stein, for coming to my aid at the last minute. This dissertation was improved greatly by her careful reading and critique.

My conversations with Dana Angluin have been invaluable; her constant efforts to get me to formalize my intuitions have been, if not entirely successful, very helpful. Many other people also contributed greatly to my experience at Yale, particularly Padmanabhan Anandan, Larry Birbaum, Dave Kriegman, Anand Rangarajan, Michael Tarr, Jeff Westbrook, and Ken Yip. I'd also like to thank my undergraduate mentors at CMU—Dario Giuse, Mark Perlin, and Masaru Tomita—for getting me started in AI and encouraging me in my research.

Michael Beetz has been the perfect officemate—his constant good cheer was a much-needed inspiration during as my writing dragged on. He also deserves credit for reading endless drafts of my writing without complaint. Many other people over the years (more than I can list—you know who you are) have also helped made my stay at Yale enjoyable and stimulating. I also thank my many students who taught me more than anyone.

None of my work could have gotten anywhere without the help and support of the departmental facility staff, who always kept things running smoothly behind the scenes. I'd especially like to thank Miriam Putterman and Suzanne Cascella for all their help over the years.

I gratefully acknowledge the support of the Fannie and John Hertz Foundation.

My sanity was preserved through the years of graduate school by my many friends in the New Haven Jewish community. I'd particularly like to thank my good friends and neighbors, Josef, Hadas, Ketoret, and Levana Chajes, for helping me to remember the important things in life. I'd also like to thank David Avigdor, Jon and Karen Berlinger, Marc and Lisa Epstein, Nahum and Michele Goldberg, Steven Lapin, Marshall Marcus, Ed and Julie Resnik, Josh and Rachel Sandman, and Frank and Paula Steen for their constant friendship and support.

To Richard Hill I owe an inexpressible debt of gratitude, for being a wonderful teacher and expanding my horizons, showing me another way.

Finally, and most importantly, I'd like to thank my parents and my sisters, without whose love, support, and encouragement I never would have gotten this far.

Contents

List of Tables

List of Figures

Passive Map Learning
and
Visual Place Recognition

Sean Philip Engelson

YALEU/CSD/RR #1032
May 1994

Chapter 1

Introduction

Led with delight, they thus beguile the way,
Vntill the blustring storme is ouerblowne;
When weening to returne, whence they did stray,
They cannot finde that path, which first was showne,
But wander too and fro in wayes vnknowne,
Furthest from end then, when they neerest weene,
That makes them doubt, their wits be not their owne:
So many pathes, so many turnings seene,
That which of them to take, in diuerse doubt they been.
—Edmund Spenser

Recall the first time you went to a particular shopping center (or any frequently visited building complex). You had a number of different places to go to, but you probably didn't know exactly where they were. So, you probably consulted maps or signs, or asked passers-by for directions. That first time, it took some time to get from place to place, and you likely made many false turns, but eventually you got your business done. The next time you visited the center it was much easier to find your way, even if you had different errands. The third time, it was even easier, and by the fourth visit, you hardly had to think about the routes you took from place to place, you just went. What happened is that you learned something about the structure of the building from the experience of walking about in it. This sort of learning has been termed *map learning*, since we might imagine that what you have learned is, metaphorically at least, a sort of mental map that helps you recognize where you are and find your way.

This dissertation investigates the problems of map learning and place recognition within the context of mobile robotics. Robots need to be able to learn maps, even in known environments, because hard-wiring the structure of a robot's environment can be very costly, and it is difficult (if not impossible) to keep such a representation up to date when the environment is changed. Furthermore, there are also cases where the structure of the environment cannot be known accurately in advance even to the system designers; autonomous mapping is certainly required in those cases.

There are many useful tasks that mobile robots can perform that require the use of some sort of environmental map. One example is that of an office or factory courier, whose function is to transfer materials between areas of a building. This task is one of the few for which mobile robots are currently being used; robotic couriers have already been installed in some hospitals, for delivering non-critical items to patients upon request. These robots contain detailed, preprogrammed maps of the buildings they work in; the hospital environments are also engineered to support reliable navigation (using infra-red beacons). Effective map-learning would cut the costs of installing such a system, both by easing initial setup, and by making the robots more flexible (they could be transferred between different buildings, say).

Another class of tasks are those in which little or no a priori information about the environment exists. One such task, ripe for robot use, is search-and-rescue missions. Rescuing people often involves going into hazardous environments, so it would be desirable to use robots wherever possible. Such rescue missions, whether from burning buildings or from caves, often require learning the structure of the environment on the fly, so that searching is done efficiently (time is usually of the essence). Also, there is no time to explore for exploration's sake (we will return to this point below). A similar task in a different domain is planetary exploration. Getting men to Mars to carry out scientific exploration and experimentation is, at present, a difficult proposition. A cost-effective solution that has been proposed is to send robots to gather scientific data. These robots will need to move about in large areas of the surface to gather that data, and hence will need some sort of mapping to support navigation.

1.1 Map Learning in Context

This dissertation claims that map learning must be studied in its context, as a resource for an intelligent agent acting in the world. Surprisingly, this has often not been the perspective of researchers in automated map learning. Rather, most researchers have focused on batch formulations of the map learning problem, in which the robot is set the single task of learning a map of its (closed) environment. There are several ways in which this kind of 'context-insensitivity' may be misguided. First, if little consideration is paid to the ways in which the map will be used, the representations chosen will be those useful for learning, but not necessarily for route planning. For example, representing 3D line segments may make it easier to build 'maps' from visual data, but will not help much with robotic route planning. Another issue is that in the noncontextual approach it is impossible to make use of the fact that robotic actions have meaning in terms of its overall goals. Only those aspects of the world which are relevant to such meaningful actions should be represented, and in such a way as to facilitate those kinds of behavior specifically. A third issue, the one which this dissertation addresses most specifically, is that of combining map learning with task-directed action.

1.1.1 Passive mapping

When you enter a building new to you, the first thing you do is not to walk about it, exploring the building completely before doing anything. Rather, you use a few common-sense heuristics (find the escalators, similar rooms are in the same area) to get where you're headed. On the way, however, you may learn something about the layout of the building, enabling you to get about more efficiently. For example, leaving the building after an appointment is much easier than finding the office in the first place. The information gained can also help you find your way in my next trip to the building.

This learning paradigm we term *passive learning*, since it does not require the learning agent (me, in this case) to actively explore its environment for effective learning. The vast majority of the map-learning literature to date has worked within the active map-learning paradigm, in which the objective is to develop exploration and learning algorithms in tandem to learn 'complete' and 'correct' models of an environment (for some combination of assumptions about the environment/sensors and definitions of 'correct' and 'complete'). The main reason that the active paradigm has dominated is that it allows a relatively simple and concise problem definition ("Learn a correct and complete map in polynomial time (with some probability)."). Using exploration also makes the learning process easier. There are, however, several good reasons to explore the passive learning paradigm.

First, as we just noted, people seem to learn maps passively for the most part. This is because we do not conform to the model implicitly espoused by the active learning camp, in that we have other things to do besides explore. Exploration and map learning is rarely an exclusive goal—rather, map learning takes place in order to facilitate the performance of other tasks. Hence, if exploration takes precedence over other activities (as it must, in the active learning paradigm), it will inevitably interfere with those tasks whose performance it is meant to support. This point of view leads us to stipulate that ultimate control of an agent's action should never come only from the needs of the learning task, but rather should be based on a tradeoff between the benefits and costs of various actions to the agent in the long term. Since the map learner can not always depend on exploration or experimentation being performed to resolve learning difficulties, we propose that the map learning system should be passive, and not depend upon controlling the agent's actions at all.

Another reason for wanting to study purely passive mapping is to consider map learning methods as lying on an activity–passivity spectrum. At different points along this continuum we have different levels of interaction between the mapping system and the agent's action-selection mechanism (whatever it may be). Current active map learning methods exist near one end of the spectrum, where action selection is subordinated to the needs of mapping. We propose instead to study the opposite end of the spectrum, where the mapping system is as independent of action selection as possible. In this way, we may lay a foundation for studying the various intermediate points of the spectrum, in which mapping and action selection interact in different ways. We have also begun investigating intermediate points of the spectrum by developing intermittent exploration methods which are layered on top of the passive mapping system, improving its efficiency. Since such exploration is not crucial to the mapper's success, the decision about whether or not to explore can be made based on the robot's current goals. In this way, some of the advantages of active mapping can be gained, without interfering with task accomplishment.

Passive mapping is hard

If we model robot actions and observations as simple atoms and the world as a graph of places or observed features, map-learning may be modeled as the problem of inferring the smallest finite-state machine (FSM) consistent with a set of observations. If the learner is allowed to be active, map-learning is often tractable. Angluin [1987] gives a polynomial-time algorithm for inferring the smallest FSM given that the learner can (a) reset the automaton to an initial state at any time, and (b) may hypothesize an automaton to an oracle which will provide a counterexample to the hypothesis, if it is incorrect. The counterexample oracle is not, of course, realistic for the robotic application. Angluin's algorithm is generalized by Rivest and Schapire [1989], who dispense with the reset in the general case by using homing sequences, and dispense with both the reset and the counterexample oracle when a distinguishing sequence is present and known (or learnable). Similar active learning techniques have been recently developed for cases with non-deterministic actions or outputs; see, for example, [Basye *et al.*, 1989] and [Dean *et al.*, 1992]. These methods use statistical techniques to learn maps to within given probabilistic error bounds.

Passive learning of general FSMs, on the other hand, has proven to be intractable. Inferring the smallest deterministic FSM consistent with a set of input/output pairs (inputs not determined by the learner) is NP-complete [Angluin, 1978,Gold, 1978]. Even inferring a machine polynomially larger than the smallest is intractable [Pitt and Warmuth, 1989]. These constitute fairly strong negative results for passive map-learning of general finite automata, particularly since the results apply to the simplest (deterministic) case; adding movement uncertainty and sensor noise only makes matters worse.

Geometry may help

The hardness results described above apply to passive learning of general FSMs with unstructured output functions. One extra source of information available to robots learning maps is geometric information about the relative locations of places. I argue that there is reason to believe that this geometric information suffices to make passive map-learning tractable.

The main reason why passive map-learning is intractable is that when state outputs are not unique, it is difficult to identify states without experimenting. When outputs are unique, inferring FSMs purely from input/output pairs becomes trivial. If states are points embedded in a plane (robot locations, say), their positions constitute exactly such an unambiguous output function. The problem is still not trivial because of two factors. First, in the real world, positions are not known with certainty, i.e, there is noise in the observation function. Second, global positions are not directly observed; rather, it is the *relative* positions of states which are observed. These two complications make the problem nontrivial, since states are no longer uniquely identifiable from single observations. However, it seems plausible that the structure inherent in the geometric outputs is sufficient to support effective learning. While there are no formal results in this area, this dissertation presents empirical results for a system which uses geometric information to passively learn maps of its environment. Formal analogues of this problem still need to be studied.

1.1.2 Cognitive architecture

Figure 1.1 depicts our picture of the high-level organization of an agent's cognitive system, showing where map-learning fits. At the lowest level, directly connected to the robot's sensors and effectors, are the *perception* and *action* systems. They implement perception and simple control routines, such as target tracking or wall following. (These ideas are developed more thoroughly in Chapter 2.) In order to achieve intelligent behavior, a *deliberator* controls perception and action. We can remain agnostic as to the nature of the deliberator. All that is required for map learning is that it somehow generate actions for the robot to perform. It could take the form of a classical planning system [McAllester and Rosenblitt, 1991], a reactive plan [Firby, 1989], or even a connectionist control scheme [Maes, 1989]. An action may be the activation of a perception routine or effector, or the formation of links between a sensor and an effector (a perceptual marker, as in "Follow that car!").

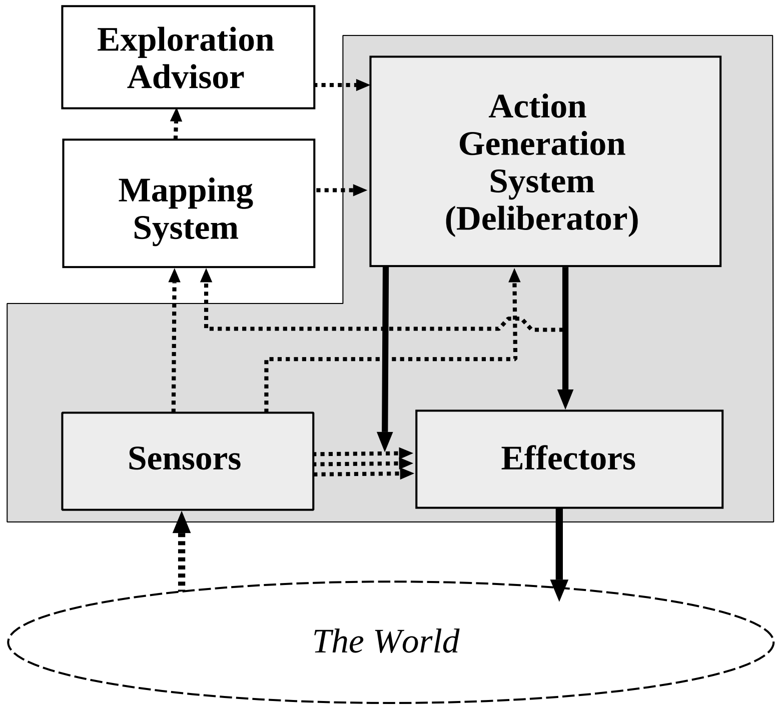


Figure 1.1: The mapping system in the context of a larger cognitive system. Thick arrows denote control flow, while dashed arrows denote flow of information.

The *mapping system* uses sensory information as well as the results of eavesdropping on the deliberator's commands to the effector subsystem to passively construct a model of the robot's environment. At present, it is not important what form this model takes. The main requirement is that the mapping system be able to support deliberation by localizing the robot and supporting route planning. The main topic of this dissertation is the construction of such a passive mapping system. In addition, intermittent exploration is implemented by the inclusion of an *exploration advisor* which follows the mapper's progress in learning the environment and suggests actions sequences to the deliberator that may aid mapping.

1.2 Organization of the Dissertation

Chapter 2 describes the abstract robot domain model assumed in our research on passive map-learning. We discuss the problems involved in domain abstraction, and propose an engineering methodology for domain design to deal with these problems. Chapter 3 describes an image-based method for visual recognition using 2-dimensional *image signatures*. The method was tested on a large database of real images, giving excellent recognition results. Chapter 4 describes a map representation scheme incorporating topological, geometric, and action-based representation elements. That chapter also proposes a unifying framework for geometric representation, and describe a novel method of abstracting experience for robust representation of behavior. Chapter 5 describes how the passive mapping system works by properly dealing with estimation ambiguity and correcting errors whenever possible. Chapter 6 discusses the results of using the passive mapping system for learning maps in several simulated environments. We also show some results on how passive mapping can improve task achievement. Chapter 7 describes how to incorporate intermittent exploration into the passive mapping framework, and shows the results of using the approach. In Chapter 8 we review related research in the field of robotic map learning. Chapter 9 sums up the contributions of the research described herein, and discusses promising directions for future research. Appendix A provides some background on interval arithmetic which is important for understanding the operation of the passive mapper.

This dissertation deals with a number of different, though interrelated, topics: passive map learning, visual place recognition, intermittent exploration, and robot simulation, among others. Since not all of these topics will be of prime interest to all readers, each chapter is written to be independent of the others, as much as possible. To that end, each chapter is preceded by a short summary of its contents, sufficient for understanding other chapters.

Chapter 2

Robot Modeling and Simulation

Summary

This chapter describes the robot model used by the mapping system described in this dissertation. Abstract modeling of problem domains is important in AI research, because it enables researchers to more easily understand the properties of proposed solutions. However, if abstraction is not done carefully, it can run up against the *relevance problem*: Solutions in abstract domains may not apply to problems in real domains. Most existing AI testbed domains (for example, the ubiquitous blocks-world) have some problems with relevance. Nonetheless, I contend that abstractions and simulation can be useful and relevant, if the right methodology is followed. Abstractions must be *justified* with respect to known properties of the real-world. Proposed solutions should be *tested* extensively against the abstractions. Then, the solutions should be used to *validate* the abstraction, by comparing the abstractions predictions of the behavior of the solution against its real-world behavior. Validation may require changing the abstraction, in which case the cycle of justification–testing–validation repeats.

Ars Magna is a simulator based on justified abstractions of a mobile robot operating indoors. There are two aspects to the model: the environmental model and the robot model. An environment consists of walls and objects. Walls obstruct the robot's motion and viewing, and define *waypoints*. Waypoints are points in the world with locally stereotypical wall structure, such as corridor intersections. Waypoints allow the

robot to accurately localize itself using local sensing and feedback control, and so will form the basis of our mapping representation. Objects are described by a set of attribute–value pairs. Objects may or may not obstruct the robot's movement or view. They may also move or change according to user-programmed rules. The robot can turn in place and move with limited translational and rotational velocity. Objects can be grasped by 'hands' and inserted into 'storage bays'. Long-range sensing is done using 'cameras', which can sense the color of walls (producing views modeled after image signatures, see Chapter 3). Cameras can also be used to find (and describe) desired objects or waypoints in the field of view. Sensing is noisy, and targets can be missed or nonexistent targets acquired. Robotic action is carried out by a set of *effector modules*, each of which implements a primitive behavior, such as following walls. Each behavior can fail in different ways, based on the failure modes of implemented robotic control systems. One important effector module is the *approacher*, which is used to approach a sensed object or waypoint. As the robot moves, odometric estimates of relative motion are accumulated. Odometric uncertainty is represented using a set-based representation, since statistical characterization of odometric errors is notoriously difficult.

2.1 Domain Abstraction

The purpose of this chapter is to describe the abstract robot model which forms the basis of the mapping system described in this dissertation. The point of using abstract, simply specified domains for research in AI and cognitive science is to focus attention on important features of some cognitive problem (or set of problems). Once the problem is stated in a simple and abstract fashion, the issues involved in its solution may become clearer. For example, much important work in planning was fueled by STRIPS and STRIPS-derived representations [Fikes and Nilsson, 1971], culminating in simple and complete non-linear planning algorithms such as those of Chapman [1987] and McAllester and Rosenblitt [1991]. In the course of this development, the shortcomings of such representation strategies for robotics also became apparent. Many features of real-world action are not adequately covered by the formalism. For example, actions in the real world are continuous, rather than atomic, and may overlap in time. Furthermore, the formalism itself entailed a particular computational architecture which was ill-suited to the task of robotic control. These kinds of problems are not specific to this example, but are inherent in the domain abstraction exercise. Since abstraction necessarily involves a choice of 'important' features to attend to, there is a constant danger that techniques developed in the abstracted domain will be irrelevant to real problems. One answer to this question is to choose domains in the real world with 'built-in' tractable formalizations, such as delivery scheduling. Domains like this do exist, and AI research has been applied to them successfully. Systems developed for such domains, however, tend to be domain-specific, with little true generality. Work on more generalized, autonomous intelligent agents cannot generally rely on the real-world being simple and abstract.

When developing intelligent agents to live in an unstructured, messy real-world environment, some abstractions and models of the real world must be constructed in order to guide research. The 'irrelevance problem' must be addressed, however; we must have some assurance that solutions to problems in our model will translate more-or-less directly to their real-world analogues. Until quite recently in AI, this question was hardly addressed. Take the example of the blocks world, as formalized in STRIPS [Fikes and Nilsson, 1971]. This provided what seemed like a reasonable formalization of the planning problem, for which good solutions were developed. However, when the planners were used to direct a real robot in a real environment, the experiment was a qualified success at best. The robot was slow and clumsy, and could not deal with more than trivial changes in the environment as it operated[1]. This was because the planner relied on assumptions in the model (such as perfect knowledge) which did not obtain in the real world. Even more recent abstract domains such as the Tileworld [Pollack and Ringuette, 1990], which attempt to address complex issues such as nondeterminism and extraneous events, have some difficulty with the irrelevance problem [Al-Badr and Hanks, 1991]. The solution to the irrelevance problem is an engineering design methodology involving three stages: *justification*, *testing*, and *validation*.

2.1.1 Justification, testing, and validation

Every domain abstraction is derived from a theoretical position, whether explicitly or implicitly. The choice of what domain features are important enough to be represented and how they are to be represented depends on the researcher's opinion about

[1] This was despite the fact that the environment was quite simple and highly structured.

how the problem should be approached. This bias is unavoidable. However, its effects can be mitigated if the model which is developed is *justified* in terms of current research in the real world. For example, the blocks world assumption that individual blocks can easily be distinguished from one another was never justified, since no vision system has yet been developed which has this ability. The assumption was reasonable at the time, but unjustified. On the other hand, assuming a vision system that can often (but not always) find doorways is justifiable, on the basis of recent research in robotic vision [Sarachik, 1989]. The reason for requiring justification is so that the abstract domain which is developed remains grounded in the real world. Strict justification is not always necessary, however. We can distinguish between three types of assumptions used to create abstractions: extrapolative, interpolative, and grounded. Extrapolative assumptions project untested ideas about the possible into the realm of future research, like the blocks world assumption mentioned above. Grounded assumptions are those which are firmly grounded in the abilities of the current state of the art, like door finding. The third method of abstraction, interpolation, examines the current state of the art and assumes abilities similar to those currently understood. So, for example, we might assume that we can build a window finder, based on our knowledge of building door finders. This sort of justification is weaker than full groundedness, but allows reasonable abstraction. Realism of error models is an important concern. Grounded abstractions can use verified models of the error modes of real systems; interpolated abstractions may use extensions of known error modes. For example, a door finder may occasionally miss doors or see non-existent doors; an assumed window finder would have similar error modes.

Once a justified abstract domain has been developed, problems may be posed and solutions proposed for them. The *testing* phase is where proposed solutions are analyzed as to their effectiveness, relative to the domain model. This testing may be analytical, as in stability analysis in control theory [Brogan, 1985], or it may take a more empirical form using simulation. The type of analysis depends on the nature of the model, the problem, and the proposed solution. The result of the analysis should, though, in all cases give enough insight into the solution to yield specific predictions about its behavior. The testing phase can be used to incrementally refine solutions until they perform well in the abstract domain.

Simply solving a problem in a model is not really enough, however. Completing the process requires *validation* of the model. This is done by implementing the problem solution for a real world instantiation of the modeled problem, and analyzing its behavior experimentally. The results of this empirical analysis are then compared to the results of the model-based analysis performed during testing, in order to examine the validity of the domain model. Those points where the model's predictions and the actual system behavior differ must be examined closely to determine the causes of such difference. Some such differences may be insignificant, while others may point towards serious deficiencies in the model. There are three possible outcomes of validation. The model may match the validation results well, and therefore be accepted as is. The results may not match completely, but some adjustments to the model may be possible in order to fix it. And finally, in some cases validation may discover fundamental problems with the domain model, such that it needs to be completely scrapped[2]. If the model is in fact retained, whether as is or modified, the cycle of testing and validation begins again with an attack on new problems.

2.2 Ars Magna

The domain developed in this chapter models a mobile robot acting in an indoor environment. The model is embodied in a simulator called Ars Magna[3]. The simulator was developed with attention to the problem of relevance discussed above. It is implemented in Nisp [McDermott, 1988], a macro package on top of Common Lisp, and consists of about 15,000 lines of code.

The primary purpose of the Ars Magna simulator was to support research in map-learning and navigation during task execution. Ars Magna's robot is a holonomic point robot, since kinematic planning is not our focus. Furthermore, the simulated robot comes equipped with built-in behavioral routines such as wall-followers. The use of such routines is justified by examples in the literature, and they allow us to abstract away from local geometry and dynamic control issues. Perception is similarly abstracted into the notion of the *view*, for recognition, and the *designator*, for manipulation. The Ars Magna environmental model consists of walls, empty

[2]Limiting the applicability of the model is a sort of model adjustment. So if the model doesn't work for some set of problems that can be well characterized, the model can still be used for research on other, admissible problems.

[3]Abstract Realistic Simulation of Mobile robots for Analyzing Goal achievement, Navigation, and Adaptation.

space, and objects (see Figure 2.1). Some areas of empty space (called *waypoints*) are distinguished by the structure of nearby walls. The robot can perceive and move relative to waypoints. They thus form the backbone of the mapping representation developed in later chapters.

This model can be justified by looking to the literature in robotics and computer vision. Few robot models have been tested and verified to date, and ours is no exception. Implementation of the model on a real robot for such testing is an area of current work, however. The details and justification of this robot/environment model form the bulk of the remainder of this chapter.

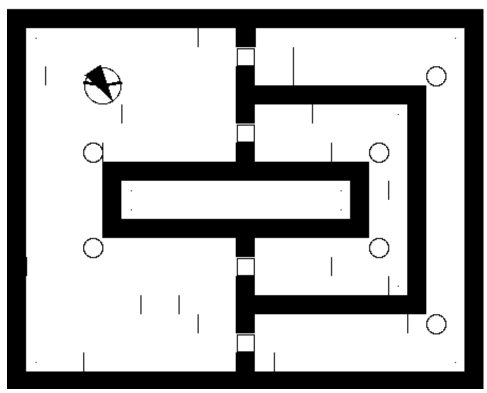


Figure 2.1: An Ars Magna environment.

Simulator overview

We will now briefly describe an example of an Ars Magna robot and its world. Figure 2.1 shows a display of an example Ars Magna environment. Walls are depicted as thick black lines. The robot is in the upper-left, heading towards the lower-right. It has two cameras, which are used for visual perception—finding and tracking objects and waypoints, as well as place recognition. Waypoints in this world are doorways and corners, indicated by squares and circles. There are a number of objects scattered about, shown by small vertical lines. In Figure 2.2 we see more detail about the robot's state. This robot has one hand which is currently empty (but can be used to hold objects as the robot moves). One of the robot's two cameras is tracking a nearby trash can, while the other tracks a doorway. The robot could decide to approach either one, using visual tracking in feedback control.

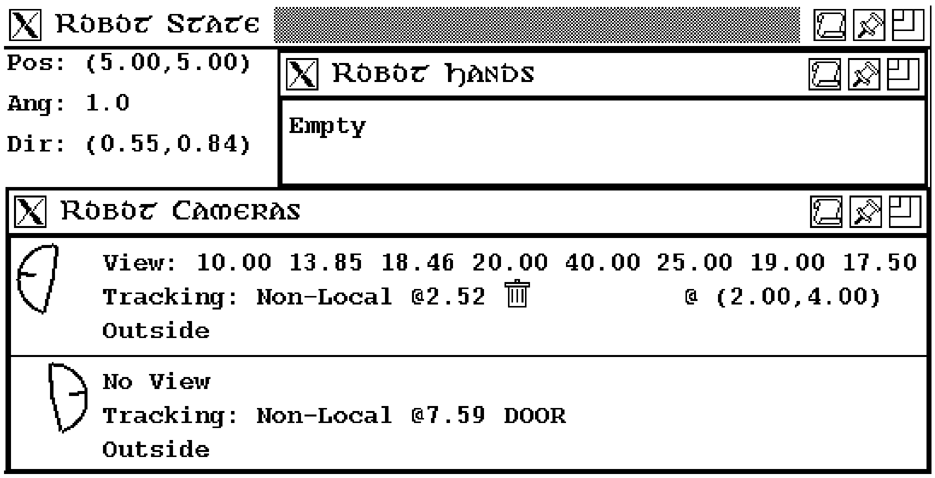


Figure 2.2: The ARS MAGNA robot display.

2.3 Indoor Environments

In order to properly model an environment, we must articulate clearly what features of that environment make it amenable to abstract modeling, and how those features may be embodied in specific assumptions about the structure of the environment. The first, most salient, feature of indoor environments that we note is that they are *path-structured*. That is, buildings are designed to direct travel along specific paths and to limit the number of distinct paths between any two points. Path-structuring is the characteristic of man-made environments that makes map-learning of the sort described in this dissertation possible. It can, in fact, be argued that the importance of path-structuring in design (documented by Norman [1980]) is due to an innate path-based mapping and navigation system in the human brain, buildings being designed in such a way as to ease human navigation.

2.3.1 Overall structure

In addition to path-structure, there are a number of more particular characteristics of indoor environments that we can consider. Most noticeable is the fact that indoor environments are 2-dimensional, for purposes of navigation. The 2-dimensional floor-plan of a single floor of a building (annotated, perhaps, with perceptual cues) provides an essentially complete navigational representation of that floor. This assumes, of course, that perception allows abstracting away objects' height. Connections between floors, then, are the only significant deviation from 2-dimensionality. The staircases and elevators which constitute these connections are generally isolated to a small set of locations on a floor. By confining vertical connections to a small set of well-defined places (such as elevator banks), 3-dimensional navigation becomes strung together bits of 2-dimensional navigation. Hence we can assume that the world is, in fact, 2-dimensional.

There are also some more detailed observations we can make about building structure. One is the fact that most obstructions have, or can be treated as having, straight edges. This leads directly to the *polygonal prism* model of the world, as assumed by Davis [1984], Moravec and Elfes [1985], and others. In this model, all objects are assumed to be polygonal prisms, with walls stretching upwards to infinity (or, at least, higher than the robot can see). This model works quite well for walls in most buildings, and reasonably well for common large objects, such as furniture. We extend this idea slightly to include a '$2\frac{1}{2}$D' model of smaller objects—they may obstruct vision, obstruct movement, neither, or both. Buildings are also often constructed on a rectilinear schema, in which walls usually meet at right angles.

Along with the fact that edges tend to be straight is the fact that they tend to be long. Most large buildings tend to consist of networks of long hallways with doors into rooms spread evenly along them. In addition to providing privacy, this sort of structure is also useful for navigation, in that paths between rooms tend to be (relatively) short sequences of hallways, rather than long, winding sequences of different rooms to go through. Fewer distinct places need to be recognized (and fewer navigational decisions must be made). A hallway also imposes a linear structure on the rooms attached to it, enabling us to say things like "the third door on the left".

2.3.2 Waypoints

All of the structural considerations discussed above are important chiefly in that they aid and enable local navigation. The last structural property of our chosen habitat is of great significance to the process of mapping (as will be seen in later chapters). This property is the existence of *waypoints*. In order to construct useful maps, it is generally necessary to discretize the world in some way, cutting it up into well-defined pieces that can be individually described and organized together into an overall map. This can be done by choosing to look at geometric landmark features as in [Leonard *et al.*, 1992], or by dividing the environment into a set of disjoint convex spaces as in [Crowley, 1984]. It turns out, however, that for highly-structured environments (such as the modern office building), environmental structure provides natural ways of performing this discretization. Kriegman's generic models for visual perception [Kriegman, 1989] use the idea that much of the structure of the environment can be described in essentially stereotypical terms. For example, we may have a generic model of a stereotypical hallway, which can then be instantiated to create a percept of a particular hallway. The model contains the constraints imposed by the generic structure of the 'stereotypical hallway'[4].

In addition to the idea of using stereotypical building blocks for describing the world, we also note the use of local *homing procedures* in mapping [Kuipers and Byun, 1988]. Suppose we could provide our robot with the ability to reliably get to a particular position from somewhere in its vicinity. Then, once the procedure has been executed, the robot can localize itself quite precisely, regardless of its original positional uncertainty[5]. We can then use these 'homing points' (or Distinctive Places, in Kuipers' and Byun's terminology) as the basis for our mapping representation. The idea of *waypoints* encompasses both these ideas, that of stereotyped environmental structure, and that of local homing.

[4]This approach to vision bears a strong similarity to Schank and Abelson's use of scripts in planning and story understanding [Schank and Abelson, 1977]. It shares some of the limitations of the approach as well, which Schank himself noted several years later [Schank, 1982]—chiefly brittleness in the face of true novelty. Recent work in causal vision [Brand *et al.*, 1993], however, maintains aspects of the generic models approach, while attaining flexibility by using true functional explanations for constructing a coherent percept of an object.

[5]This is essentially the same idea as that developed by Erdman and Mason [1986] in their study of robust manipulation strategies. They show how manipulation strategies can be developed which result in a known final state which need know nothing about the initial state of the system. Our methods differ, though, in that we allow sensor feedback to attain stability.

The main idea behind waypoints is that there are point-like locations in the environment whose local structure is recognizable and distinguishable from its surroundings. For example, a doorway is (ideally) distinguished from its surroundings by the presence of doorposts, a lintel, and a door (open or closed). Such a location we call a *waypoint*, as it forms a significant localization point for the robot as it goes on its way[6]. We group waypoints which share a generic structural description into *waypoint types*, each of which describes a class of recognizable points in the world.

Because of the complexity of the world, the variety of possible sensing modalities, and the multifarious interactions between them, we will not try to devise an abstract description language for waypoint types. Such an attempt would almost certainly not be expressive enough to contain all distinctions we would like, so we define waypoint types procedurally[7]. There are two main robot procedures which, if implemented, may define a waypoint type. The first is a *recognizer*, which is a perceptual routine that decides if the robot is at a waypoint of a given type or not. The second routine is the waypoint type's *approacher*, which implements a local homing procedure for waypoints of the correct type. This means that there is some vicinity around each such waypoint such that if the approacher is executed with the robot in that area, it will move to the waypoint. If the robot is not in the vicinity of a waypoint of the appropriate type (or is blocked in some way from reaching the waypoint), the approacher should fail. Approachers may be implemented modularly, as finding and tracking routines combined with a general feedback controller, although specialized servo routines may also be developed (as in [Sarachik, 1989]). Naturally, our model allows for both recognizers and approachers to make errors; details of these error modes are discussed in Sections 2.4.2 and 2.4.4 below. A third procedure, not fundamental but nonetheless useful for each waypoint type, is an *orienter*. The function of an orienter is to determine the robot's orientation when at a waypoint, relative to a frame of reference local to the waypoint. For example, an L corridor intersection uniquely defines a local coordinate frame by its two arms. An important problem in mapping is determining the robot's orientation, and the use of waypoint orienters helps, as discussed in Chapter 5.

[6]Informally, we also note that useful waypoints will be those that the robot *must* pass through to get from place to place. Again, we may speculate that the abundance of such points in human-designed environments demonstrates the importance of this notion in human mapping.

[7]It should be understood that we do not thereby eschew the idea of a declarative waypoint type representation. Such may indeed prove possible, but is beyond our scope here.

Procedural representation

A few words are now in order about the notion of defining waypoint types in terms of robot procedures. When we say that a particular set of procedures represents the waypoint type 'doorway', we mean only that the set of positions that the procedures pick out in the world (by recognition and approach) corresponds roughly to what we recognize as doorways. No further meaning is intended, and the use of the word 'doorway' is intended only as a mnemonic. As such, we must beware of the fallacy of 'wishful terminology' [McDermott, 1981a] and remember that the true waypoint type denoted by an approacher/recognizer pair depends on a complex interaction between the procedures and the robot's environment. The reason we can use such mnemonics at all is the fact that the procedures are designed to pick out, say, doorways; if they are designed well, they should usually do so. If at times they do not, and 'hallucinate' doorways where there are none, we let the robot treat such hallucinations as 'doorways' for its purposes (provided the hallucinations are consistent). This does preclude including deeper contextual information about waypoints' relationships to their surroundings, since the extension of the 'doorway waypoint type' may differ from the 'Platonic' doorway.

There are, however, certain restrictions on the sorts of procedures that can function as approachers, recognizers, and orienters. Let us consider an arbitrary waypoint type with an approacher A and recognizer R. We will discuss the ideal behavior of these procedures, ignoring sensor noise and errors for now. The first requirement we place is a consistency condition that requires that R return true whenever A has just been executed successfully. We next require that the points picked out by A as waypoints are sparse. That is, we consider all positions in the environment which the robot could end up at after a successful run of A, and require that no two of these positions are closer than some minimum distance. This ensures that waypoints can be distinguished from one another, at least in principle. The third constraint that is needed for approachers is that each waypoint position p picked out by A is surrounded by a dense set (p's *approach region*) within which A will bring the robot to p. These idealized requirements suffice for our purposes to pin down the concept of a waypoint type. When we consider uncertainty below, some other requirements will be necessary to preserve the ideal behavior of approachers and recognizers, but such considerations are not fundamental to the basic concept of waypoint type. Also note that in practice it will not be possible to rigorously prove that these properties hold for a given waypoint type; they will have to be verified empirically.

2.3.3 Objects

Our approach to modeling objects in the world is a fairly standard attribute-value approach. Each object is defined as a set of values for a set of predefined attributes (for example, `color = green`). Attributes may be continuous (size) or discrete (color). Users may define attributes specific to particular domains, but a set of basic attributes integral to the Ars Magna system are predefined. The first is `obstructs-movement`, a boolean, which denotes whether or not the robot can walk over the object. Similarly, `obstructs-view` denotes whether or not an object can be seen through (or over). These two attributes are independent, so one can have, for example: file cabinets (obstructing movement and viewing), glass panels (obstructing movement but not viewing), curtains (obstructing viewing but not movement), and pieces of paper (obstructing neither movement nor viewing). Two inherent attributes of an object are its `size` and its `weight`. An object's `size` is a measure of how wide it is (height is ignored). Perceiving `size` is not too difficult (provided a good estimate of depth is known), but estimating `weight` is clearly harder. This is modeled by different noise parameters of these attributes (see Section 2.4.2). All objects are conceived of as potential containers, so they all have a `capacity` attribute. If an object's `capacity` is non-zero, it may contain other objects, up to a total combined size of the `capacity`. Containment may, of course, be recursive. In addition to the physical attributes described above, objects may also have two 'perceptual' attributes, `color` and `signature`. `Color` is a categorization of the object's color into symbols such as `red` or `green`. The `signature` attribute is a numerical view signature (as for walls, see Section 2.4.2 below) giving an object's appearance in a view, if it obstructs vision.

By modeling objects as sets of attributes, we are actually modeling the perceptual abilities of the robot. A more realistic approach might model lower-level physical properties of objects—their shape perhaps, and reflectance spectrum. Perceptual routines would then be functions from these 'physical' properties into percepts (with appropriate noise models). For our purposes, though, the effort simply is not worth it. The question comes down to how fine-grained a model is needed for sufficient realism. In Ars Magna, different attributes are modeled at different levels of detail.

Obstruction of view and movement by an object depends on the geometry of the object (position and size), because that can affects the robot's large-scale behavior. Color, on the other hand, is only used for recognition in our work, so more detail than a single symbol would be wasted.

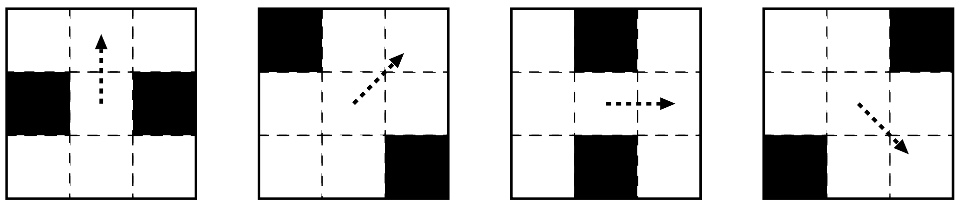


Figure 2.3: Doorway waypoint schema. Filled squares denote walls, empty squares denote space, and arrows denote the direction considered to be 'through' the door.

2.3.4 Implementation

In addition to the justifiable abstractions described above, the actual Ars Magna implementation uses some other reasonable, though non-realistic, abstractions in order to simplify implementation. The most obvious is a grid approximation for walls. The world is quantized on a grid, and each grid cell is either filled or empty. (The grid only serves to define walls—the robot moves fully in 2-dimensions, not on the grid.) Filled cells represent walls. Truly diagonal or curved walls are thus impossible, but since most angles in buildings are right angles, this does not seem unreasonable. Waypoints are represented by categorizing grid cells. Hence if the robot is anywhere in a waypoint cell, it is considered to be in a 'doorway' or 'L-junction' or whatever. Cell categorization is determined by 'waypoint schemata', which decide waypoint types automatically, based on local wall structure. For example, a doorway is defined as an empty cell with walls in two opposing directions, and space elsewhere. This schema is depicted in Figure 2.3. This way of determining waypoints is somewhat unrealistic, since in reality nearby objects may affect waypoint categorization[8]. It forms a reasonable first approximation, however, given a model of designation errors such as that described below in Section 2.4.2.

[8] For example, Sarachik's door-finder occasionally mistakes the space between a water fountain and the wall as a doorway [Sarachik, 1989].

2.4 The Robot Model

The class of robots we consider are point robots which can turn in place and move with limited velocity and angular speed. Collisions are detected by the cessation of motion. Robots can be configured with a number of 'hands', each with a limited strength. A hand can grasp a single object nearby the robot whose weight is less than the hand's strength. Force sensors can tell if a hand is grasping something. Hands cannot coordinate to pick up a larger object together. Hands can also be inserted into containers; we assume hands take up essentially no space. Long-range sensing is performed by a set of 'cameras'; sensing modalities are described further below. Each camera can face in a different direction relative to the robot. Cameras can also be inserted into containers—this stretches physical plausibility, but consider cameras as being mounted on small arms which can be maneuvered to look into open containers. The robot can also be equipped with a set of built in 'storage bays', each with a limited capacity. They can be opened and closed by the robot. Hands and cameras can also be inserted into these bays.

More important than the physical robot model is our model of low-level robot control, which is what allows us to abstract from the details of the physical world. We base our analysis on the emerging paradigm of *hybrid control systems*; see [Firby, 1989] and [Gat, 1991]. In this view, low-level control is handled by a set of primitives which produce behaviors such as wall-following or grasping[9]. Given a set of such primitives, a *deliberator* can reason using abstract models of the behavior of the robot. There are a number of interesting issues to be explored in the design of such primitives, some of which are discussed below. One important point is that, assuming such primitives exist (and work as advertised), most kinematic and dynamical constraints need not be modeled for high-level reasoning. This is because such details are taken care of at the lower level. If a corner cannot be turned due to kinematic constraints, we expect the 'corner-turning module' to detect that fact and either find a way around the problem (if possible), or else fail cognizantly. Thus we can develop a robot model (and mapping system) that abstracts away from most kinematic concerns.

[9]These modules by themselves, can be thought of as comprising a subsumption architecture [Brooks, 1986, and see below]. We allow, however, for high-level reasoning to tinker with the operation of behavior control, as discussed later on.

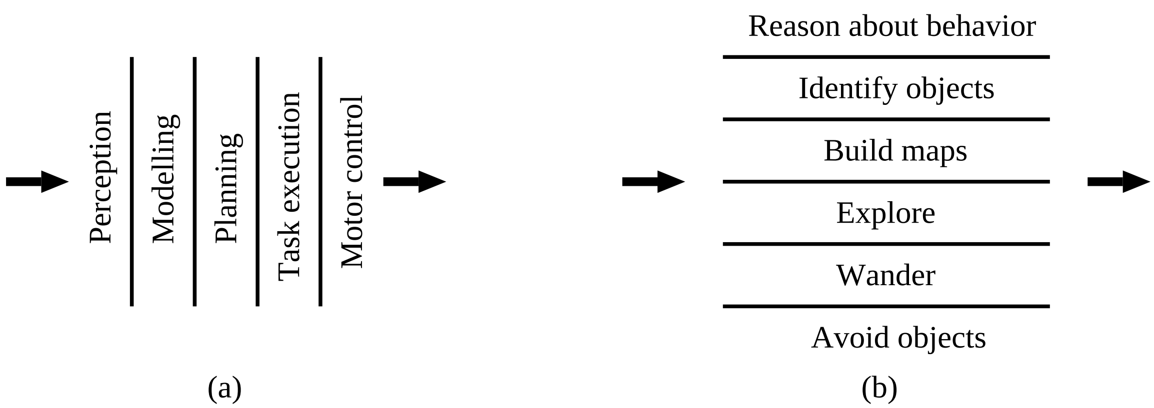


Figure 2.4: Architectures for robotic control (after Brooks [1986]). (a) Traditional sense-plan-act architecture. (b) Behavior-based architecture showing task decomposition.

2.4.1 Behavior-based control

How should a robot's behavior be controlled? The classical answer has been that a massive sense-think-act loop is used (depicted in Figure 2.4(a)). In this paradigm, perceptual routines figure out the structure of the world, the robot cogitates in order to form a complete plan for its goals, and then it executes the plan. While this model provides a convenient modular decomposition of the robot control task, it has failed to produce robust working robotic systems. An alternative modularization is based on the notion of *behavioral decomposition*. In this view, modularization is 'horizontal'; control modules are developed for each of a set of primitive robot behaviors, such as obstacle avoidance and wall-following (see Figure 2.4(b)).

There are several good reasons to work within this paradigm, noted by Brooks [1986]. First is the fact that the robot often has several proximate goals, such as obstacle avoidance and target tracking, that it needs to achieve at the same time. Since these goals are typically (mostly) independent, it makes sense to modularize along task lines and build separate modules to achieve each such goal. Furthermore, each behavior will only need a subset of the total available sensory information, so complete world modeling is not necessary. Related to this is the fact that different behaviors may need to deal with sensor characteristics differently, since the effects of errors may differ. A fourth point is that a system composed of a number of modules, each with its own competency, will be robust—if one module has trouble achieving its goal, other modules may still be able to proceed. Finally, this approach lends itself to easy extensibility; to add new robot capabilities, just add a new behavior module.

Hybrid behavioral control

Purely behavioral systems are quite good at performing 'continuous', short look-ahead tasks such as "Follow the corridor without bumping into anything, while looking for a doorway." However, they are less well suited to more 'discrete', large-scale tasks such as those traditionally addressed by AI planning. Such tasks typically require the synthesis of a plan containing sequences of qualitatively different actions (eg, moving vs. grasping), which are difficult to express as the continuous action of a constant set of behaviors reacting to their environment. Even more difficult is incorporating complex global flow control ("if you see a door, do A,B,C, else do E,F,G"). For these reasons, recent research has focused on so-called 'hybrid' systems, in which instantaneous control is vested in a behavior-based subsystem, and larger-scale issues are handled by a deliberative planning/execution subsystem. We will generically term this system the *deliberator*, making no assumptions about its internal workings. The deliberator controls the robot by adjusting the parameters of the behavioral system: activating/deactivating behaviors, setting behavior parameters, and so on. In this way, specific sets of behaviors can be activated in a desired order, sequencing the robot's activity.

Behaviors also communicate with the deliberator by sending it signals. In particular, since designing behaviors which never fail is impossible, we require that, whenever possible, they fail *cognizantly*. That is, when a behavior fails (eg, a wall follower loses the wall), it must be able to give some diagnosis of what sort of error occurred. This is needed so that contingencies can be planned for and dealt with intelligently. In addition, as we will see in Chapter 4, knowing what sorts of failures occur aids us in remembering behavior for later recall.

Arbitration

The *behavioral control* approach (as it is called) has proven quite successful in the design and implementation of real robots getting about in the world. One key problem that behavioral decomposition gives rise to is the *arbitration problem.* Given that low-level control is implemented as a set of independent behaviors, how are their decisions as to proper robot control to be combined into a single control signal for the robot? This is an extremely important question, since poor arbitration may lead to none of the behaviors' goals being achieved, due to interference between them.

Subsumption An early approach to behavior arbitration was the *subsumption architecture* of Brooks [1986]. In a subsumption architecture, behavior modules are constructed out of smaller computational units, which compute such things as a desired direction of travel, or whether a collision is imminent. Units in higher-level behavior modules can then override, or *subsume*, outputs (hence inputs) of units in lower-level modules. For example, a goal-tracking behavior's desired direction would subsume that of a wandering behavior. While the subsumption architecture has proven to be useful in the development of robotic systems (see, eg, [Connell, 1989] or [Mataric, 1990]), designing subsumption control systems is still something of a black art. This is because the subsumption architecture does not explicitly provide arbitration; rather, arbitration must be built into the behaviors themselves. For example, the wandering behavior described by Brooks contains a component called "avoid", which uses internal state of the avoid obstacles behavior in order to wander while avoiding obstacles. The need for such gyrations severely reduces modularity, which was one of the main goals of the behavior-based approach in the first place. In addition, this makes it difficult (if not impossible) to analyze the overall behavior of complex systems in terms of their behavior, since different arbitration methods may be used between different behaviors.

Channels In his dissertation, Gat [1991] describes a somewhat more general and organized approach to arbitration than subsumption. One of the main reasons that the subsumption architecture does not easily support modularity is that signals coming from new behavior modules may inadvertently suppress a critical signal in other modules. Gat addresses this problem by encapsulating meaningful values (both effector signals and intermediate values) as *channels*. A channel may be thought of as a device taking an arbitrary number of inputs and arbitrating between them to compute an output. Gat considers simple channels, such as maximum channels, averaging channels, or prioritized channels, but more complex arbitration functions may be built out of several such primitive channels. The advantage here is that the arbitration strategy may be tailored to the type of value under consideration, avoiding the above-mentioned problem with subsumption. Furthermore, arbitration is isolated in the channels, ensuring some level of modularity. However, the channel approach still requires great attention on the part of the system designer to ensure that behaviors can work together. Addition of a new behavior may still cause problems if the channel which receives its output has an inappropriate arbitration function. In some sense, it would be desirable to have a global arbitration strategy, which would treat all behaviors equally, allowing the design effort to go into creating the individual behaviors.

Potential fields Probably the simplest uniform arbitration scheme is based on *artificial potential fields*. An artificial potential function (as introduced by Khatib [1985]) is a function from the workspace of the robot to the real numbers, with the idea that the robot is to minimize the potential of its location. Thus, we might consider behavior modules as causing the robot to follow the gradient of some potential function defined over the robot's environment. That is, the robot always moves downhill with respect to this function. For example, the potential near obstacles is set high, so that they will be avoided. We may combine the potentials of several behaviors by adding them together, resulting in a potential that includes the 'goals' of its components. The complete potential field need not be computed, though; if each behavior module produces a desired velocity vector (corresponding to the gradient of a potential field), the vector sum of all the vectors is the gradient of the total potential. This approach has been used to good effect, for example in Arkin's implementation of motor schemata [Arkin, 1989].

The main difficulty with the potential summing method is the occurrence of local minima, points where the potential is minimized (so the robot is motionless) but which do not correspond to the actual goal. This is because the potential field method takes an essentially local view of the situation. A potential field method that does not incur local minima is that of Koditschek [1989]; however his method is global and cannot be used as an arbitration mechanism. A second, related, difficulty with potential field approaches is the fact that they can sometimes produce absurd trajectories. In his dissertation, Slack [1990] gives an example where a potential summing technique applied to the goal of driving the robot straight down a corridor would drive the robot through a doorway. This also is due to the local, knowledge-poor nature of potential-summing arbitration.

Constrained optimization An approach that allows more flexibility than the potential summing method is the use of *constrained optimization*. In this view, each behavior module may be thought of as computing an expression of the utility (to the module) of every robot action. Utility functions can often be modeled as simple functions of the deviation of the robot action from some nominal preferred action. The potential summing approach may be viewed as a particularly simple method of this sort; the utility of a velocity $\vec{x}$ to a module with gradient $\vec{x}_i$ would be $-\|(\vec{x} - \vec{x}_i)\|^2$. Multimodal utility functions may also be used, for example for an obstacle avoidance module which is agnostic about which direction to take around an object. An implementation of this kind of optimization approach is applied to mobile robot navigation by Payton [1986]. This approach often allows the system to avoid the absurd results produced by potential field approaches, by having behaviors supply the arbitrator with more acceptable solutions, allowing a more satisfactory compromise. Another optimization approach is examined in [Engelson, 1990], which explicitly incorporates directional information into the objective function. This allows deliberative processing a hand in arbitration, for example specifying that the robot should go left around an obstacle[10].

An enhancement to simple optimization is proposed by Yamauchi and Nelson [1991]. In their system, each behavior module outputs a set of constraints, specifying which actions the behavior forbids. The robot then chooses an action within the allowed set to perform. Incorporating hard constraints like this into an optimization arbitration scheme has real advantages. For example, an obstacle avoidance routine would rule out moving into an object, regardless of other modules' preferences. The best *feasible* action would then be chosen. Similarly, potential summing's problem with the doorway in the hall disappears if a constraint is placed not to go through the doorway.

[10]This insight is due to Slack [1990], who uses it to develop a local navigation algorithm using environmental models.

Rosenblatt and Payton [1989] discuss a very general constraint-posting scheme, in which modules can specify sets of constraints to be satisfied; the arbitrator then finds a value which satisfied as many of the constraints as possible. The authors get around the NP-completeness of general constraint satisfaction by restricting their attention to conjunctive constraints.

Reactive planning

The notion of a plan directing activity via a behavior-based low-level control architecture leads us away from classical conceptions of what a plan is. It is no longer sufficient to conceive of a plan as a linear sequence of atomic actions, since behavior takes place over time, and more than one goal can be pursued at once. Furthermore, information is constantly coming in (possibly due to changes in the world) as the robot goes about its business, so the plan should be flexible enough to use this information. Agre and Chapman [1988] make the point that a plan should, in some sense, be viewed as 'advice' to an action-generating system; it should constrain behavior to be within certain broad limits, rather than specifically directing it. A slightly different approach with the same motivation, is the idea that plans should directly control action, but do so *reactively*. The formulation of reactive plans is addressed by Firby [1989], where he proposes the idea of Reactive Action Packets, or RAPs. A RAP is a collection of methods for achieving a goal, annotated with contingency information, such as when each method is applicable, or under what conditions the RAP should fail. RAP methods invoke either other RAPs or primitive actions, which may be carried out in parallel. The RAP system is built around a complex interpreter which determines what RAPs should run when, depending of annotated preconditions, constraints, and priorities. The sequencing layer of Gat's ATLANTIS architecture [Gat, 1991] uses the RAP paradigm, extending the notion of primitive actions to apply to a behavior-based substrate. Each behavioral module has three primitive actions associated with it: initiation, termination, and monitoring. The former are clear; monitoring examines signals coming from the behavior to check on its state. Such information may then be used by the monitoring RAP to adjust its behavior.

Some recent work which builds on many of the ideas of the RAP system is McDermott's RPL language [McDermott, 1991]. Rather than encapsulating a plan in separate, independent packages like RAPs, RPL conceives of a plan as a high-level *robot control program.* RPL is a programming language, much like Lisp, with added constructs for building parallel reactive plans. All the usual features of parallel programming languages are included, with the addition of *fluents* for interprocess communication. A fluent is a sort of active value, which can be monitored or waited on. So, to control robotic behavior, a RPL program invokes behaviors, and then monitors fluents which the behaviors set in order to communicate with the program. Another important RPL concept is the *policy*, which is an action constraining other actions. For example, a delivery robot may have a policy, "Always look for boxes," to help it carry things. Policies can also serve as soft protections which actively maintain a needed state of affairs for a subplan. In general RPL is somewhat more expressive than the RAP system, though RAPs allow bits of a plan to be more easily thought of as independent entities.

Effector modules

In our formulation, behaviors are implemented by *effector modules.* An effector module is a black box that achieves or maintains a spatio-temporally local state of affairs. By "spatio-temporally local" we mean that events far from the robot in time or space are not considered relevant. For example, an obstacle-avoidance module maintains the state of not colliding with obstacles, and a doorway-approacher achieves the state of being in a doorway. The reason for requiring spatio-temporal locality is so that effector modules need not maintain a lot of internal state to achieve their tasks. Gat [1993] eloquently argues that use of internal state in autonomous robots should be used only for 'abstract' high-level reasoning, but not for direct robot control. His main point is that for stored state to be useful, it must enable the robot to make accurate predictions about the world. However, long-range predictions are only likely to be accurate if they abstract away from minute detail. I can more easily predict that I will be in my office tomorrow afternoon, than that I will be typing the letter "i" at 2:39pm. Another reason for restricting internal state in effector modules is that it makes it easier to reason about their behavior. Consider a module containing no internal state whatsoever, constituting a pure transfer function from sensors to effectors. Whenever the module is active in the same position in the same environment, the same behavior will ensue, regardless of the robot's recent history. In particular, this means an effector module can be interrupted (so that the robot can do something else), and then later restarted, as long as the robot is at the same place. This property will be seen to be crucial for representing behavior so that it may be repeated (Section 4.5).

Effector modules may be activated or deactivated at any time by the deliberator. When a module is activated, it may be passed parameters which specify its behavior more precisely. So, a wall-follower may be parameterized by the desired robot speed. In addition to specifying the values of particular quantities, effector parameters may also connect different modules. For example, a grasping module may use the outputs of a cup-tracker and a hand-tracker to accomplish its task. Here we will only need to consider a particular kind of connecting parameter, the *designator*. In brief, a designator is a data structure that refers to a particular object in the vicinity of the agent. This reference exists by virtue of the designator containing the perceptual information needed to find or otherwise manipulate the object in some way (see Section 2.4.2). A designator can connect a tracking module (which keeps the designator up-to-date) with a behavioral module (such as an approacher, or a grasper).

Information also flows the other way, from effector modules to the deliberator. First of all, when an effector module terminates, it emits either a success or a failure signal. Success is signaled if the module determines that its goal has been achieved. Maintenance modules, on the other hand, only terminate when they fail. Failures are accompanied by an error code, denoting the type of failure that occurred; this is used for run-time error-recovery. Success and failure messages are instances of more general *effector signals* that can be sent from effector modules. Each signal consists of a *label* and a (possibly empty) set of *values*. Success labels are the symbol `SUCCESS` and failure labels are symbols denoting different error codes. So, a wall-following module may fail with the error signal `LOST-WALL` if it lost track of the wall it was following. Other types of signals are also possible. Modules may signal when they take control of the robot, as in the case of a tracking module which signals when it has begun to track a designator. Other signals may signify important perceptual events that an effector module looks for. Our wall-following module may, for instance, look for doorways in the walls it follows; when a doorway is encountered, signals like `DOORWAY-ON-LEFT` may be generated by the wall-follower. When such a signal is received by the deliberator, a decision may be made to stop following the wall and go into the door (or, it may not). We can also think of finding routines as effector modules which signal a designator value, which can then be used to connect other tracking and movement modules. In Chapter 4 we will discuss how the robot can repeat its behavior, by translating lists of effector signals into robot plans.

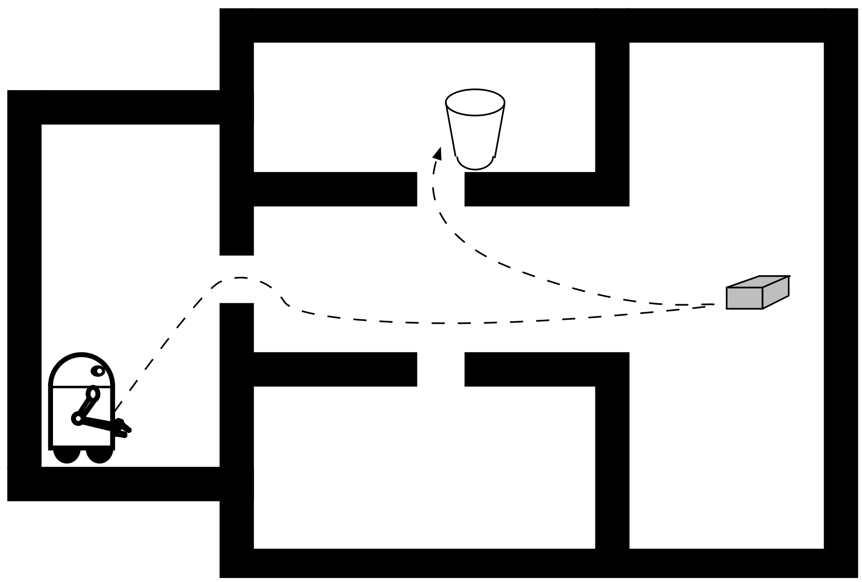


Figure 2.5: A simple delivery task.

Example 1 *As an example, take the fetch-and-deliver task depicted in Figure 2.5. The robot first looks for a doorway, using its cameras. It then moves straight for the doorway and goes through. It then proceeds to follow the corridor and looks for a package. When it finds and gets to the package, it grasps it in one of its hands and turns around in place. It then looks for another doorway, goes to it and through. It then finds the receptacle, inserts the hand with the package into it, and releases the package.*

2.4.2 Perception

Perception can be divided into two general categories: *referential* and *non-referential*. Referential perception is that which attaches reference between data structures and things in the real world. For example, an object recognition routine connects an object model with its referent in the real world, if all goes well. We speak here teleologically, so if perception fails and 'recognizes' an object which isn't there, we can still consider the 'ideal' perceptor as creating reference. Not all perception is referential—some useful types of perception process sensor readings to compute results which do not refer directly to things in the real world. This result may be an abstract descriptor used for recognition of a world state, or a preferred action[11]. An example might be a perception routine mapping sets of sonar readings to probably safe robot headings. In this case, the heading does not correspond to a thing in the world; rather, the sensor readings are associated with a particular behavior pattern. Non-referential perception tends to be more task- and domain-specific, since it makes less reference to the 'real world'.

Designation

Referential perception is encapsulated in the concept of *designation*. A *designator* is a data structure which contains information about the relation between the robot and a thing in the world enabling the robot to behave relative to that thing. In this way, a simple visual marker acts as a designator, since it enables the robot to servo to an object's location. A more complex example would be an instantiated geometric object model giving enough information for the robot to grasp something. Since such information quickly goes out-of-date if not tracked, designation is a sort of short-term memory of the robot's surroundings. As other researchers have noted (particularly see [Agre, 1988], [Agre and Chapman, 1987], [Firby, 1989], and [McDermott, 1990]), designation (or *indexical-functional* representation) allows plans to refer to objects teleologically, by their function in the plan. This eases many aspects of plan construction, placing the burden of resolving reference on perception. This makes planning considerably easier—if I want a glass of water, I don't have to worry

[11] An interesting example of non-referential perception is found in the relationship between drumming and dance in African and Caribbean music. Dancing in this tradition is not representational and symbolic as in Western dance such as ballet. Rather, the dancer listens to the drumming, and dances in a beat which interacts with the drums; conversely, the drummers watch the dancers, and create rhythms which talk to the rhythm of the dancing. Since the music makes use of polyrhythm (multiple overlapping time-signatures), a change of perceptual focus can change one's entire perception of the music. It is the dancer's perception of the beat that informs the movements and rhythm of the dance; there is no explicit referent of this percept, it is rather associated with the action of replying to the rhythm. "In African music, excellence arises when the combination of rhythms is translated into meaningful action; people participate best when they can 'hear' the rhythms, whether through understanding or dance." [Chernoff, 1979]

about which glass to use beforehand; I just go into the kitchen, look for a clean glass, and use it. A designator, by definition, instantiates the needed functional aspects, in a form that can be directly used by effector modules. Planning about complex real-world situations can then be carried out on an abstract plane, based on functional considerations and high-level models of effector behavior.

Waypoint types

Recall the definition above of waypoint types as locally distinguishable, stereotyped bits of environmental structure. We decompose waypoint approaching into the perceptual processes of finding and tracking and the effector process of servoing (also described above). These perceptual processes are referential (referring to a particular waypoint), so we first discuss acquisition and use of waypoint designators. To make the discussion concrete, we will focus throughout on the example of a doorway.

A perceptual model of a waypoint type consists of four routines: a *finder*, a *tracker*, a *recognizer*, and an *orienter*. A finder for a waypoint type T is a perceptual routine which detects locations of type T from afar. Finding may rely on many different sorts of sensory inputs—doors may be detected as depth discontinuities by sonar, as rigid quadrilaterals in the image plane, and so on. One good example is Sarachik's doorway detector, which uses vision to find vertical edges surrounding depth discontinuities [Sarachik, 1989]. Finding a location of a given waypoint type provides a designator which gives the direction of the waypoint found from the current location of the robot. The waypoint type's tracker enables the robot to track the designator once found. Complex model-based tracking is probably not often necessary, since feedback control using image-based feature tracking has proven to be quite effective [Hager *et al.*, 1994]. If the robot then decides to go to the designated (and tracked) location, a simple servo routine will usually suffice[12] (see below).

A recognizer detects when the robot is at a location of type T. One example of waypoint type recognizers is the sensor abstraction work of Kirman *et al.* [1991]. A sensor abstraction is a mapping from a low-level sensor space (eg, sonar readings) to a smaller, high-level sensor space (eg, corridor intersection type). An abstract sensor

[12]We assume routine obstacle avoidance is taken care of by the effector control system. See, for example, [Brooks, 1986] or [Connell, 1989]. Tracking waypoints with occlusion is eased by the fact that they don't move.

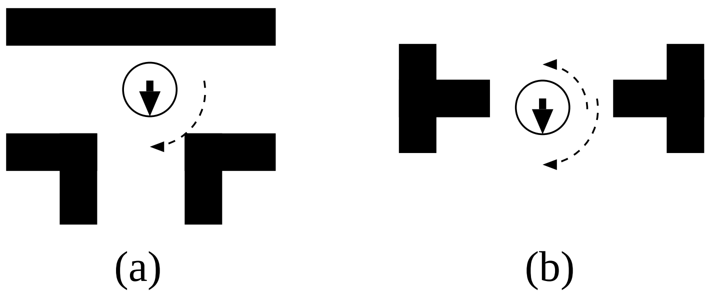


Figure 2.6: Orienting with respect to a waypoint. (a) T-junction and its local coordinate frame. (b) Doorway; note the two possible coordinate frames that can be oriented to.

can be treated the same as any other sensor. In particular, the authors use abstract sensors as inputs into a Bayesian network [Pearl, 1988] for robot control. This reduces the dimensionality of the input space, so that the conditional probability matrices in the Bayes nets are manageable. The mapping that the authors used for junction categorization was designed heuristically, using two stages—one which mapped raw sonar readings into categories 'short', 'medium', and 'long', and a second which used these 'categorical' sonar readings to determine junction type. The abstract sensor thus developed was 'trained' by running the robot around corridors, tweaking the mapping based on its performance. Almost 90% accurate recognition was achieved with this relatively simple method.

Finally, we assume that the environmental structure at a typed location allows the robot to locally orient itself relative to its surroundings. Consider a T corridor junction. We may reasonably assume that, given proper sensors, a robot could determine with some accuracy its orientation in a junction-relative coordinate frame (see Figure 2.6(a)). Some waypoint types may also have some orientational ambiguity, however—consider a doorway (refer to Figure 2.6(b)). Here, the inherent symmetry of the situation allows two possible orientations, 180° apart. Hence, we stipulate that a waypoint type's orienter gives a set of possible robot headings in the local reference frame. Thus the doorway orienter will always give two possibilities, for example, 32° and 212°. Resolving this ambiguity requires context, and takes place at a higher level, when these sensor readings are integrated with the map.

Apart from general errors in designation (discussed below), there are several kinds of noise peculiar to waypoint sensing. First is recognition error, in which a recognizer returns an incorrect result. The model we use here is a simple one which specifies two probabilities: the probability that a waypoint type will be recognized where there is none, and the probability that a waypoint type, where one exists, will be recognized incorrectly (or missed). An example of an error of the first type would be the robot deciding it was at a doorway when in the middle of a room. An error of the second type might be the robot deciding it was at a doorway when really in a T intersection. A more realistic model would specify the conditional probabilities $P(T_1|T_2)$ of returning a waypoint type T_1 given that the robot is at a waypoint of type T_2[13]. For the purposes of this dissertation, however, the simpler model suffices since the mapping system does not explicitly use the details of the probabilistic model. Orientation can also be corrupted by noise. As for odometry, we assume that the noise is bounded with known bounds. This is reasonable since, if necessary, repeated sensing can be used to drive outlier probabilities down. In any case, this is an improvement over previous models used in mapping, where absolute orientation was assumed known (as in [Kuipers and Byun, 1988] and [Mataric, 1990]).

Object perception

As with waypoint types, non-local object perception includes finding and tracking, which are largely top-down. In addition, though, there is a third function, *description*, which gives more information about perceived objects than is supplied. Designator acquisition is a combination of both finding and description; the designator which is returned may contain previously unknown object attribute values. The other difference between object and waypoint perception is the use of local designation. While only long-range behaviors are meaningful relative to a waypoint designator, objects can be manipulated locally. However, a view of an object at a distance rarely gives enough information to permit manipulation when the robot nears the object. Hence, we require the robot to acquire a *local* designator in order to manipulate an object, even if it already has a non-local designator. This is an over-simplification of a more realistic scheme which would treat all the sorts of information needed for different sorts of manipulation separately. However, it forms a reasonable first approximation.

[13]Even more realistic would be a model which conditioned probabilities on environmental structure directly; however constructing such a model would be prohibitively complex.

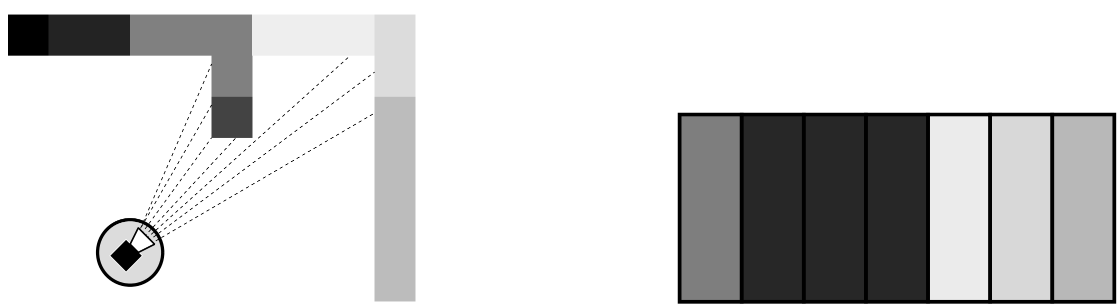
Designator acquisition starts with the top-down specification of the attribute values of the object to be looked for. Categorical attributes are specified as disjunctions of possible values, as `(:color (:red :green))` specifies that an objects color should be either red or green. Numerical attributes can be specified as a specific value, as in `(:size 5)`, or as a range, as `(:size (3 8))`. The more attributes are specified, the more likely it is that a matching designator will be found, since more specific knowledge can be brought to bear. On the other hand, depending on the noise present in the estimation procedures for various attributes, specifying more attributes may make it less likely that the specification will be matched.

In addition to the top-down specification of desired objects, description requires bottom up sensing of unconstrained attributes. The simplest model of this assigns a *notice probability* to each attribute, giving the probability that a value can be determined for the attribute without it being constrained. That is, it may be possible to estimate a property of a sensed object, even without top-down constraints. We can also take into account the fact that specifying certain attributes makes sensing others more likely. For example, we may be able to estimate the capacity of a container more easily given its size. Hence, we allow attributes to *support* one another. Saying that a_1 supports a_2 means that if a_1 is constrained in a sensing operation (so can be determined more easily), the chance of figuring out a value for a_2 is increased. We simplify matters by assuming that all supporters support an attribute by the same amount. Call the base probability of noticing an unspecified attribute a, p. If a is to be noticed, either a is noticeable on its own, or it is noticeable by virtue of one of its supporters. If we assume supporter independence, the probability of noticing a is modeled as $1 - (1 - p)^{n+1}$, where n is the number of a's supporters specified. As the number of specified supporters goes up, the probability of noticing a goes up. This model could also be generalized so that attributes can have variable levels of support.

When a sensing operation is attempted on an object, the object's real attribute values are perturbed according to attribute-specific noise parameters. Discrete attributes are corrupted by having some probability that the sensed value will be chosen uniformly from the universe of all possible values. Continuous attributes (such as size) have Gaussian noise added to them. Noise increases with distance, up to some maximum. Before an object description is checked against a potentially matching object, the relevant object attributes are noise-corrupted; if they still match the description, the object is considered a match. If the object matches, each of its attributes unspecified by the description is checked as above for being noticed. If an attribute is noticed, its value is corrupted according to the noise model above, and returned.

Designation errors

In addition to description noise, designation itself may be erroneous. First, both false positives and negatives are possible when acquiring designators. A false positive occurs when an object is seen that does not exist, due to shadows or specular reflections. False negatives are more common—objects are not seen, either because of attribute noise, or because they are too far away. There are also two types of tracking errors we can consider. The first is the unexpected loss of an object track, either due to occlusion or due to one of a variety of imaging phenomena (shadows, accidental alignments, etc.). The second sort of tracking error, not modeled in Ars Magna, is transference of a track from an object to a similar-looking one in the field of view. This can happen when the lines-of-sight to the two objects coincide as the robot or the objects move.



(a) (b)

Figure 2.7: Sampling to get a view. (a) Plan view of the robot and walls. (b) The robot's view.

Views

In addition to waypoint and object perception, we model a third form of perception, for place recognition. The objective is to determine the robot's location, assuming that it is at one of a finite number of known locations. This only requires computing some 'signature' from sensor inputs which can be reliably matched to memory. The model used here is based on the image-based place recognition method described in Chapter 3. An image can be represented as an array of values (an *image signature*), computed by applying a *measurement function* to various subimages. Each value in the array represents some overall visual property of part of the scene. This is pretty abstract already, but we go one step further and use 1-dimensional signatures, since we assume that the essential structure of the world is 2-dimensional. Pieces of wall are assumed to have a 'signature' attribute, which is a number representing some sort of measurement (say, of texturedness). We term our 1-D signatures *views*. Views can be used both for simple place recognition as well as orienting the robot.

There are other ways of defining views that may be modeled (with more or less fidelity) in this way. These other notions of views may take into account geometric properties of the environment. One simple way is based on the method of Atiya and Hager [1991] for computing correspondences between observed vertical lines and those in a stored model. They project the lines into the groundplane to get a set of 2D points, and then compute the parameters of triangles of points (which are invariant to viewpoint). These triangles can then be matched to stored triangles and a consistent correspondence derived. With this method, a set of triangles can be considered a view, view matching becomes triangle correspondence, and robot relative orientation can be determined.

A view is computed by finding the signature of the nearest wall at angles sampled across a camera's field of view (see Figure 2.7). Some objects also occlude vision, as discussed above, and are treated as walls when computing views. The object's `signature` value determines how the object will appear in a view. Noise is modeled by a contaminated bounded uniform distribution, where an observation of the true signature s has probability p of being chosen uniformly from within ϵ of s, and probability $1 - p$ of being chosen uniformly from the range of all possible signature values.

Example 2 *Now let us consider the types of sensing used in the fetch-and-deliver example above. The robot first looks for a doorway waypoint type; the designator obtained is used to servo the robot into the doorway. The robot then proceeds down the hallway, looking for the package, described as, say:*

```
((:shape :parallelepiped) (:size (1 4)) (:color (:brown :grey)))
```

When a non-local designator is acquired, about the middle of the hall, the robot goes to it. Though it is not used in this example, the returned designator contains a sensed description of the object, something like:

```
((:shape :parallelepiped) (:size 2.1)
 (:color :grey) (:obstructs-view :no))
```

The robot must then acquire a local *designator on the package, in order to grasp it. After grasping, the robot proceeds to the next door in the same way as the last. When the robot arrives at the doorway to the repository, it might get a view from the doorway to check that it is in the right place. Otherwise, it might put the package in the wrong receptacle. The receptacle is found and approached using a non-local designator, then a local designator is acquired so that the package can be deposited.*

2.4.3 Odometry

A third sort of perception, different from the others discussed above, is odometry. This is the computation of relative robot movement from measurements of the motion of its wheels. As might be expected, this computation is fraught with error. Errors arise because of quantization errors, because of miscalibration, and because of wheel slippage. The effect of the last is the greatest, and also the hardest to deal with, because it is extremely difficult to model. This is both due to the complexity of the physical situation, and because slippage depends on the character of the floor the robot travels over. Arkin [1987, Ch. 7] developed a noise model for odometry based on slippage effects, which includes several noise parameters:

Translational loss Variance of odometry for purely translational motion, as a function of distance traveled;

Drift Variance of rotational odometry when translating, as a function of distance traveled;

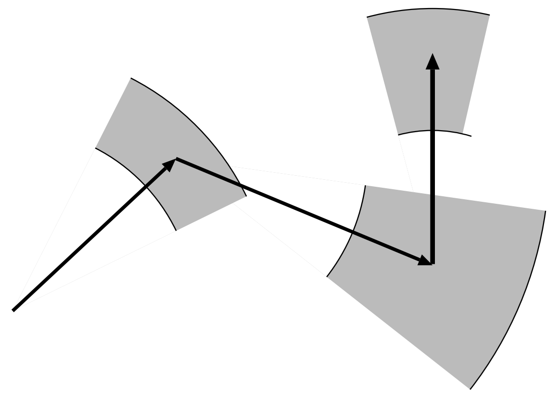


Figure 2.8: Odometric uncertainty at each step of a robot trajectory. The uncertainty of the three-step composite transform has a complex and unwieldy shape.

Rotational loss Variance of odometry for purely rotational motion, as a function of angle turned;

Skitter Variance of translational odometry when rotating, as a function of angle turned.

He found that this model could be used derive confidence bounds on odometric error. However, use of this method required the painstaking gathering of data on the various floor surfaces that the robot was to be run on. In addition, the robot had to know at all times what kind of surface it was running on. Until it is possible for the robot to automatically calibrate its odometric model to the floor surface, Arkin's detailed modeling approach remains impractical. We thus must use some kind of worst-case approximation to model odometric error.

There are two main approaches we may use for describing odometric noise: statistical and set-based. The statistical approach assumes that noise occurs according to some known probability distribution. Usually, the distribution is assumed normal, and the variance parameterizes the amount of noise. The set-based approach, on the other hand, does not assume any distribution on noise, but rather assumes known bounds on the noise. Thus from an observation we can compute a set to which the true parameters must belong (according to the set-based model).

Noise is commonly assumed to be Gaussian, because this assumption leads to simple estimation procedures such as the Kalman filter [Gelb, 1974]. For many problems, this assumption is close enough to reality to produce reasonable results. Simple distributional assumptions like this are not valid for odometry, however, because of the causal complexity of odometric noise. Even if slippage effects could be modeled as normal, the effects of slippage on different wheels propagate through the nonlinear kinematics of the robot, destroying normality. But even modeling a single wheel's slippage as normal is impossible, since slippage is affected nonlinearly by the speed and angular velocity of the robot and by the variable friction of the floor surface. For this reason, we use a set-based modeling approach, assuming that we can place worst-case bounds on odometric error, while remaining agnostic as to the actual error distribution. We allow for the possibility of outliers (as they *will* occur), but assume the probability is low. If we decompose a trajectory into a sequence of small segments, each composed of a rotation and a translation, we can parameterize the uncertainty over each segment by percentage bounds on translational and rotational error. This gives rise to a sequence of wedge-shaped regions (Figure 2.8), which when composed produce an uncertainty region for the entire trajectory. We approximate these wedges by intervals, in order to simplify computation (see Chapter 4 and Appendix A). The end result, then, is an interval of possible changes in position since the last odometric reset. This representation can take into account direction-sensitive errors; if the robot travels in a straight line, then forward uncertainty will be high, while transverse uncertainty will be low (if rotational uncertainty is not high).

2.4.4 Movement

There are two sorts of effector modules: *maintenance* modules and *achievement* modules. Maintenance modules seek to maintain some relationship between the robot and its surroundings, hence should only terminate on failure. An example of a maintenance module is a wall-follower (such as Connell's [1989]), which seeks to maintain a fixed distance from the wall, and keep the robot moving forward at a fixed speed. The wall-follower fails if it loses the wall (violating the first constraint) or if the robot is unavoidably blocked by an obstacle (violating the second constraint).

An achievement module attempts to bring about some state in the world, for example, the robot being in a doorway. Achievement modules succeed when they terminate in the required state (as far as the module knows) and fail if they determine that the state cannot be achieved. So, a door-approacher would fail if the door was lost from sight, or the robot was unavoidably blocked. In fact, as noted above, only a general approaching module may be needed, since designators (tracked by perception modules) provide the state information needed to servo the robot to a desired location. A second important type of achievement module aligns the robot with its current waypoint. Each waypoint type defines a local reference frame (or several, if symmetric) in relation to the wall structure that defines the waypoint. The robot may then align itself with such a coordinate frame through feedback control. Waypoint alignment is important for angular stabilization of the robot, as we will see when we discuss mapping. Without some sort of stabilization, orientation errors tend to increase without bound, making the robot's position impossible to estimate.

Example 3 *To summarize the fetching example, let us consider the activation of effector modules in the scenario above. The first module activated is a sensing module, which looks for the doorway. It produces a designator, which is then passed to two modules: a tracking module which keeps track of the designator as the robot moves, and an approacher module which is active until the doorway is reached. The wall-following movement module is then used to follow the wall; at the same time, an object sensing module is used to scan for the package. At some point, the package is found and the approacher takes over again. When the package is reached, a grasping module is used to take hold of the package. The rest of the scenario is similar.*

2.5 Analysis

While our focus in this chapter has been on the development of a robot model and simulator to support research on mapping and navigation, the framework we have devised appears to be somewhat more general. As noted in the introduction to this chapter, there is a real need for realistic testing domains which are still abstract enough to be tractably dealt with. The abstractness criterion may be evaluated operationally, by seeing if useful research is engendered by considering the domain. Realism, however, may be considered to have two components. First is groundedness, which, as discussed above, requires justification, testing, and validation to be complete. A second important property a domain must have is that it must exercise the limits of the research it is intended to support. That is, we recognize the fact that research is driven by the problems addressed and the ways in which they are represented (note the prolific influence of the STRIPS formalism), therefore we should choose our domains with care, so that the problems and representations they engender will advance the field rather than, at worst, spawning a series of highly technical trivialities. To this end, the remainder of this section will analyze how Ars Magna relates to the primary issues of concern to modern planning research.

2.5.1 Simulator design

When designing a simulation testbed to support work on planning, there are a number of issues of that must be addressed. There are three issues of *verisimilitude* which describe the ability of a domain to probe the limits of planning methods. First off, the world must be sufficiently *complex* so that interesting problems may be posed. Second, the world must not be completely *controllable* by the robot; there must be some uncertainty in the outcomes of its actions. And third, the world should not be completely *observable*; questions of sensing and reasoning about knowledge are finessed away in perfect-information domains. These requirements all stem from the desire to evaluate planning systems realistically—the real world is complex, incompletely controllable, and incompletely observable. We chose to model a mobile robot in an indoor environment, justifying our model using current research in robot sensing and control. This choice satisfies the criteria above well—the world is both geometrically and ontologically complex, robot actions may fail in realistic ways, and sensors are local and noisy. One ancillary requirement stemming from real-world conditions is that control over time properly lies in the simulator, not in the planner; ideally they should be implemented as separate processes. The Ars Magna simulator is implemented using processes in Lisp, providing asynchronous planning and action, with time controlled independent of of the planner.

From the standpoint of the experimenter, the simulator must support *variability*, so that complex experiments may be designed to elucidate particular features of planning systems under consideration. We distinguish two types of variability: *flexibility*, such that a wide variety of different test problems may be posed, and *tunability*, where various system parameters may be changed within a particular test scenario (eg, sensor noise). Ars Magna provides flexibility in several ways: world structure can be specified, objects may have user-defined attributes, and manipulation methods may also be defined. This all allows arbitrarily complex scenarios to be constructed, within a uniform simulation framework. Ars Magna also provides a set of tunable parameters, controlling sensor and effector noise, action speed, and robot dynamics. These parameters correspond to physically meaningful concepts, and so may be varied meaningfully during experimentation.

2.5.2 Issues in planning research

Examining current areas of planning research allows us to identify specific issues that should be addressed in a simulation domain for planning. Several of these have been identified by Al-Badr and Hanks in their critique of the Tileworld domain [Al-Badr and Hanks, 1991]. The analysis below is abstracted, to some extent, from their discussion. They identify the following current research problems in planning: exogenous events, the time cost of planning, richness of world models, sensing and effecting, measuring plan quality, and multiple agents. Our work currently addresses only the first four of these issues. The question of measuring plan quality is, as Al-Badr and Hanks note, properly the concern of the planning researcher—'goodness' is not built into the world. Support for multiple agents is conceptually simple in the Ars Magna framework, and will probably be incorporated in the near future. The remainder of this section discusses these issues in more detail and summarizes how Ars Magna addresses them.

Exogenous events

The problem of exogenous events is how an agent deals with events in the world not under its control. Other than the fact that such events occur, we can identify three important qualities that exogenous events may have in a given domain. The first is *complexity*—are the events structurally simple (eg, tiles appearing) or complex (eg, prey hiding from the agent). The second quality is *variability*—how many different sorts of events are possible? It is conceivable to have a domain with only one event; regardless of that event's complexity, great planner flexibility need not be required. Thirdly, we wish events to be *inhomogeneous* and treat different parts of the world differently. An example of a homogeneous event is the periodic 'wind' in the NASA Tileworld [Philips and Bresina, 1991]. Homogeneous events are easier to deal with, since their effects on different objects are similar. Ars Magna supports exogenous events through *kickers*[14], intermittently applicable procedures that may move or change objects in arbitrary ways. Kickers are implemented as Lisp functions, and so inherit the full complexity and variability of Lisp. Direct support is provided for common operations, such as moving objects around. Since kickers are applied (usually) to specific objects, inhomogeneity is assured.

Handling time

There are three issues to be addressed as far as time handling goes: independence of simulator time from thinking time, synchronization between the world and the planner, and the ability to 'skip ahead' and speed up the simulation. To achieve 'temporal independence', we use multiple threads of execution; simulator time proceeds as a sequence of 'ticks', in each of which a quantum of action takes place in the world. Synchronization is achieved by having robot actions call planner-supplied continuations when important events occur. This method is open to abuse, due to its flexibility (arbitrary bits of code can be executed in the simulator process space), but detecting this is easy, so a modicum of discipline suffices to keep things clean. Time skipping is not supported in Ars Magna, but it is easy to conceive of it being implemented by increasing the action quantum. This would have the disadvantage of reducing the temporal resolution (and hence the accuracy) of the simulation, but that problem is inherent when skipping ahead in a simulation (unless an exact closed-form solution is available, which it is not here).

Model richness

In analyzing the richness of a world, we ask the following questions. Are many different sorts of objects possible? Furthermore, are these different objects related to each another in interesting ways (eg, categorization or functional interaction)? And can complex causal mechanisms be devised, either in terms of particular objects changing over time and via interactions with the agent or the world (eg, eggs + mixing + frying → omelet), or in terms of devices built out of other objects (eg, chassis +

[14] Term due to Drew McDermott.

engine + transmission + wheels → car)? Ars Magna, as noted previously, supports an unlimited variety of objects in the world. Conceptual relations between objects are properly in the agent's mind, in terms of the relations between object attributes. So, a bird is an object with wings and feathers (say). Functional relationships can be established by world designer-defined manipulations—for example, the ability to lock a door is a manipulation that relates a key to the door. Kickers can be easily used to define internal causal properties of objects. Composite objects are not currently supported, but are not conceptually difficult in our framework.

Sensors and effectors

As we consider the sensors and effectors by which the agent interacts with its world, we must consider four issues: bandwidth, noise, cost, and interface cleanliness. By bandwidth we mean the amount of information the agent can gather or change about the world, compared with the information actually there. For example, a color-blind robot would not fare well in a world in which color was a functionally important cue. Sensors and effectors must also be noisy to be realistic; we should like the noise parameters to be adjustable in a meaningful way for experimentation. Furthermore, sensing and effecting cost the agent, at least in terms of time. Such costs must be part of the domain model, and again, should be adjustable. And finally, for purposes of controlled experimentation, the interface between the agent and the world must be clean, well-defined, and documented. Without this requirement, it is hard to be entirely clear about what an experiment really measures. We have attempted to address all of these concerns in Ars Magna. Objects in the world are defined by their location and attributes, all of which can be sensed and modified (through manipulations). In fact, Ars Magna's object sensing models a combination of both top-down and bottom-up sensing, allowing complex perceptual interactions to be specified by the designer. Other sensors are provided for uncertain odometry, coarse perceptual matching and for categorizing types of environmental structures. Range sensors are not currently modeled; however, for map learning this is not a severe loss. All sensors and effectors have associated noise and error modes, all of which can be adjusted by the experimenter. One of our most important system design goals was that the agent-world interface be clean and conceptually simple. The interface is documented in detail in [Engelson and Bertani, 1992]. All sensor/effector operations are specified similarly, using a small set of predefined data types when complex information is communicated. User-defined manipulations can be defined in a structured manner, and are treated similar to built-in operations.

2.6 Related Work

The robot model and simulator described in this chapter is a member of a large confraternity of AI testbed domains. In recent years, a number of complex domains have been developed for research and testing of planning and learning systems. Most of these domains have been inspired by robotics, though they range from the fairly well-grounded to the highly abstract. In this section, we will discuss some of the more prominent of these systems, in terms of the requirements discussed in the last section.

2.6.1 Tileworld

A recent effort whose goals are similar to ours is Pollack and Ringuette's Tileworld simulator [Pollack and Ringuette, 1990]. The Tileworld domain is a game played on a grid, where the robot's objective is to push square 'tiles' into polyomino-shaped 'holes'. Holes have limited capacity, requiring reasoning about limited resources. Further, tiles, holes, and obstacles can randomly appear and disappear, providing some uncertainty in the world. Various parameters of the system may be adjusted via 'knobs', allowing for controlled experimentation. The Tileworld is, however, fairly simple and controlled, as well as completely observable (the robot is omniscient), as Al-Badr and Hanks [1991] point out, hence it is hard to say whether it supports addressing fundamental issues of world complexity, incomplete knowledge and prediction, and real-time planning.

2.6.2 Seaworld

Another simulator built as a testbed for AI system experimentation is Vere and Bickmore's Seaworld [Vere and Bickmore, 1990]. The simulator was developed for use in testing their integrated AI agent, whose goal is to explore issues of combining techniques for planning, natural language comprehension/generation, knowledge representation, and episodic memory. The agent controls a robotic submarine which receives natural language commands over a simulated 'radio link'. The environment is reasonably complex, with a wide range of different objects with which the agent can interact. Further, full 2D geometry is used; the robot does not live in a discretized space. However, this complexity is not easy to control, thus making it difficult to perform controlled experiments for comparative analysis. Experimentation appears to consist primarily of 'anecdote analysis'—seeing if what the agent does in a few particular scenarios makes sense. Also, like the Tileworld, action and perception are error-free, finessing a crucial issue in real-world systems.

2.6.3 Truckworld

The Truckworld domain developed by Firby and Hanks [1987] satisfies most of our criteria well. The simulator provides for exogenous events via update methods that can change objects after each robot action. Time is handled by skipping the simulator ahead by a number of time steps after each action; unlike the Tileworld, however, the duration of each action is determined by the simulator, not the planner. Simulated environments are graphs of places connected by roads, which further points up the atomic action model. Other than generic visual scanning (Is there an object in my vicinity?), no true action parallelism exists. The Truckworld simulator is implemented using an object-oriented methodology, which allows for unlimited model richness, by allowing the user to write their own methods for object manipulation and evolution. Agent resources are also integrated into the world—movement uses up fuel. Sensing and effecting are error-prone with adjustable error-rates.

The Truckworld approach to time has some difficulties, both of implementation (update methods must reason about different lengths of time explicitly) and of operation, in that actions are essentially atomic (nothing happens during them), though they may take some time. The main difficulties with the Truckworld are in using it for controlled experimentation. There is little structure for defining and customizing objects in the world, due to the general nature of the object-oriented methodology. This makes it difficult to establish systematically qualitatively vary an experimental setup. Effector error is limited to arm clumsiness, speed variability, and movement failure; it is difficult to vary these parameters in a systematic fashion. Furthermore, it is hard to relate these to generic physically meaningful concepts. The generality of the object system makes it difficult to systematically vary perceptual error, since errors are hidden inside object-specific sensing methods.

2.6.4 The Phoenix environment

One of the most elaborate simulation environments used today for AI research is the Phoenix environment (henceforth 'Phoenix'[15]) developed at the University of Massachusetts [Cohen *et al.*, 1989]. Phoenix simulates forest fires in Yellowstone park; the focus of the AI research is on developing agents to fight these simulated forest fires. The simulator was designed to support research into real-time adaptive planning, coordination of multiple agents, knowledge representation for plans, and distributed planning. Phoenix is a discrete-event simulation, which uses physically-based models to simulate the spread of forest fires and actions of firefighting bulldozers. The individual agents have knowledge limited by their sensory capabilities. The Phoenix world is quite realistic as well as tunable by the experimenter. It is also instrumented and interfaced with a statistics package for ease in running and analyzing experiments. It is an excellent model of what we are trying to achieve for indoor robotics with Ars Magna.

[15]Properly speaking, 'Phoenix' refers to the whole system comprising the simulator and the agents operating within it, however for clarity we refer to just the simulation component by this name.

Chapter 3

Image Signatures

Nihil est in intellectu
quod non primus in sensu.
—Aristotle

Summary

This chapter proposes an image-based place recognition technique, using arrays of measurements called image signatures. An image signature is an array of measurement values. The input image is tessellated into a grid of subimages. Each value in a signature is derived from a subimage by a predefined measurement function. Measurement functions usually map from image regions to numbers or angles. A measurement is meant to represent a coarse physical property of a portion of the scene. By representing images at a large scale, signatures achieve significant data reduction, and also afford stable recognition. Reduction of images to signatures can also be made very fast, since only simple image-based processing is required. When used with appropriate measurement functions, image signatures support efficient recognition over a range of viewer positions and orientations.

Signatures are matched against one another to confirm hypotheses of the robot's location, and can be used to efficiently index into a knowledge base to generate such hypotheses. Recognition is decided by element-wise matching. Signatures from several measurement functions can be used together for conjunctive match filtering. Small viewpoint rotation and

translation can be corrected for by matching signatures at offsets and at different scales. Signatures can be indexed for hypothesis generation using standard techniques for multidimensional point indexing. These techniques provide efficient search methods for finding good matches for an input image. Image signatures thus provide a simple, efficient, and effective basis for visual place recognition.

An inherent difficulty with any image-based recognition method is the *recognizability problem*. This problem consists of two factors. First, many scenes give little information for localization (eg, blank walls). Second, there are too many scenes in the world to remember them all—which should be used for recognition? The image signature paradigm provides a direct approach to dealing with this problem through the notion of distinctiveness metrics, real-valued functions of signatures such that high values indicate unambiguous scenes. As for recognition, distinctiveness values can be combined for different measurement types for added reliability. Optimization techniques are then used for directing camera motion to find local distinctiveness maxima. Such distinctive viewpoints will tend to be unambiguous. Furthermore, since there are few local maxima, relatively few scenes will ever be used for recognition. This further underscores the usefulness of the image signature method.

3.1 Introduction

Any map-learning system must be able to tell if it is at a place it already knows about (whose description it should update) or an unknown place (and therefore augment its map). Thus, place representations must include a description of the place sufficient to support reliable recognition. Some of this burden can be laid on representations encoding the relative positions of places in the map (see, for example, [Davis, 1984] and [Smith *et al.*, 1986]). Chapter 4 describes the *diktiometric* map representation scheme, which integrates both topological and high-level geometric knowledge in a uniform framework. In any case, due to odometric uncertainty, perceptual cues must be used as well.

3.1.1 Sensor-based place recognition

Local sensor-based place recognition can be quite difficult, because often the features easiest to recognize are common to many similar places, and hence different places cannot be easily distinguished. This means that it is easier to decide that one is not at a particular place than that one is at that place, thus local place recognition should be conceived of as a filter on possible candidate places. Not all such filters are created equal, of course, and we can evaluate them by two criteria: *stability* and *distinguishability*. A stable recognition method is one whose results do not change if the robot moves slightly. This ensures that the method is reliable. Distinguishable recognition methods are those which can tell the difference between different places, even if they appear superficially similar. For example, measuring the shape of an office will usually be less distinguishable than examining its contents.

In this chapter, we are interested in exploring place recognition methods using vision, because its high information bandwidth makes distinguishability a reasonable goal. One approach would be to attempt to reconstruct the geometric structure of the scene, perhaps extracting and recognizing objects, and matching the extracted structure to those of remembered places. The reconstruction problem is, however, very difficult. Fortunately, full reconstruction of the scene is probably not necessary for place recognition. We are not interested in recognizing visible objects *per se*, but rather in effectively filtering the set of possible places that we may be at. We argue that this can be done using image-based methods, wherein simple 2-dimensional representations of input images are used to filter possible place matches for recognition. This approach avoids the difficulties inherent in reconstruction, as well as providing distinct computational advantages.

3.2 Image Signatures

We need a representation of an image which can be stably matched to representations of other images showing the same scene. It should also be insensitive to the presence of image noise. To satisfy these requirements, an *image signature* is defined as an array of measurement values, each value derived from a portion of the original image. An input image is tessellated evenly on a grid (typically square), and a *measurement*

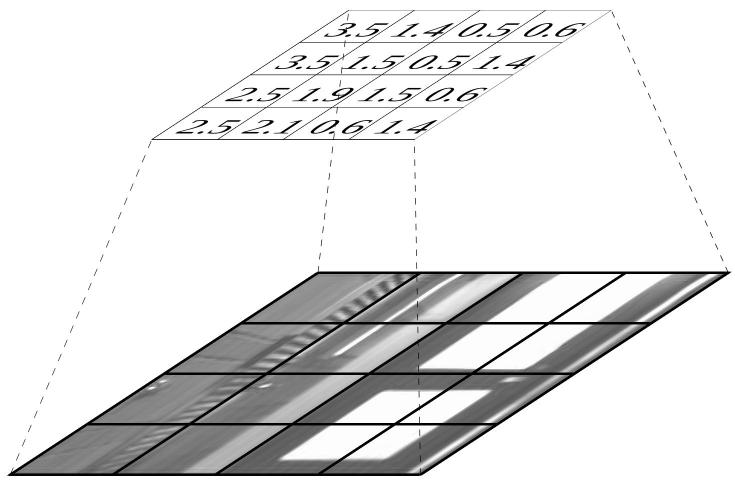


Figure 3.1: Derivation of an image signature.

function is applied to each subimage, giving a value for the corresponding position in the signature array. Such measurement functions map from image regions onto a low-dimensional codomain, typically either the reals or a small discrete space. One example of a measurement function is the dominant edge orientation (DEO) for a region (see Figure 3.2); it is reasonable to expect that two image regions having a similar dominant edge direction share some physical similarity. Signature matching is done by comparing corresponding elements for similarity; there are different types of matching, discussed below. A large number of images can be stored this way, since typical signatures will take up no more than perhaps a hundred words in an optimized implementation.

The remainder of this section develops the machinery necessary to use image signatures for place recognition. We shall begin with a discussion of how signatures are computed. This amounts to a discussion of measurement functions. In addition to the particular measurement functions developed during this work, there are general evaluation criteria that can be used to design good sets of measurement functions in general. Section 3.2.2 describes signature matching methods, including how to deal with viewpoint motion. We then describe how signatures derived from multiple measurements are used together for place recognition. The section concludes with a discussion of efficient hypothesis generation methods.

3.2.1 Measurement functions

A crucial part of a place recognition system based on image signatures is the choice of measurement functions. This choice is, to some extent, dependent upon the robot and its environment. Although there is, as yet, no well-founded theory to guide development of good measurement functions, there are some heuristic principles that can be used to design and evaluate measurement functions. Also, measurement functions should be quickly computable. Measurement functions map image regions into values; without loss of generality, we will speak of measurement functions applying to entire images.

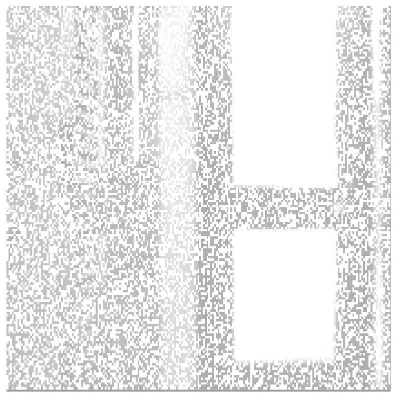
When considering measurement functions, it is often the case that we need to distinguish between *sparse* and *dense* measurement types. Sparse functions measure an overall property of an image region by extracting a set of tokens from the region and computing an aggregate measurement for the token set. A sparse measurement value can occasionally change drastically if the viewpoint changes so that a token moves into or out of view. By contrast, dense measurements compute a value by averaging a pixel-based measurement over the image region. Thus small viewpoint changes produce small changes in dense measurements.

Some measurement functions

We have developed a set of measurement functions which work reliably in our building. They are described below. Examples of signatures derived using the various measures are shown in Figure 3.2. While there are, as yet, no formal results describing the properties of these functions, empirical evidence for their usefulness is given in Section 3.4.1.

Dominant edge orientation

The first measurement function we shall discuss is the dominant edge orientation (DEO) in a region, suggested by Nelson [1989] for image-based homing. The intuition behind this is that since edges are discrete and mostly derive from scene features (occlusions, albedo, and so on), the dominant orientation in a subimage will usually



Image

$$\begin{bmatrix} 0^\circ & 180^\circ & 0^\circ & 180^\circ \\ 90^\circ & 0^\circ & 0^\circ & 180^\circ \\ 0^\circ & 180^\circ & 0^\circ & 180^\circ \\ 90^\circ & 180^\circ & 0^\circ & 180^\circ \end{bmatrix}$$

DEO

$$\begin{bmatrix} 3.5 & 1.4 & .55 & .57 \\ 3.5 & 1.5 & .55 & 1.45 \\ 2.5 & 1.9 & 1.5 & .57 \\ 2.5 & 2.1 & .57 & 1.4 \end{bmatrix}$$

SGD

$$\begin{bmatrix} .39 & .34 & .16 & .20 \\ .36 & .38 & .17 & .18 \\ .37 & .40 & .23 & .22 \\ .37 & .43 & .22 & .25 \end{bmatrix}$$

ZC

$$\begin{bmatrix} .30 & .29 & .08 & .13 \\ .32 & .18 & .11 & .15 \\ .23 & .14 & .11 & .18 \\ .20 & .11 & .14 & .19 \end{bmatrix}$$

TEX

$$\begin{bmatrix} 66 & 113 & 103 & 163 \\ 67 & 88 & 121 & 140 \\ 47 & 44 & 90 & 121 \\ 36 & 35 & 96 & 120 \end{bmatrix}$$

ESTR

$$\begin{bmatrix} .92 & 1.1 & 1.3 & 1.2 \\ .84 & 1.0 & 1.2 & 1.1 \\ .83 & .92 & 1.0 & 1.0 \\ .79 & .86 & 1.0 & 1.0 \end{bmatrix}$$

RI

Figure 3.2: Examples of image signatures for various measurement functions. All angles are given with 0° vertical. SGD is given in multiples of 45° (the scale that the matcher uses).

change little with motion, but will be useful for recognition. DEO is calculated by convolving the image with the two first-order 3×3 Prewitt operators (see [Ballard and Brown, 1982]) to estimate the image gradient at each pixel. Attention is then restricted to those pixels whose gradient magnitude is both a local maximum in the gradient direction, and which is above a threshold, $\vartheta_{\mathrm{grad}}^{\mathrm{DEO}}$. The selected pixels are then classified into one of eight categories denoting the edge orientation implied by the gradient. The orientation to which most of the pixels in the subimage are assigned is the DEO value of the subimage. Since DEO extracts edges, it is a sparse measure.

Significant gradient direction

Another measurement, closely related to the DEO, is the significant gradient direction (SGD). The directional intensity derivatives at 0, 45, 90, and 135 degrees are estimated using rotated versions of the 5×3 Prewitt operator (at skew angles, the convolution masks are 5×5). The mean of all pixels above a threshold, $\vartheta_{\mathrm{der}}^{\mathrm{SGD}}$, in each derivative subimage is computed, giving a 'response' for each direction. The average of the top two directions is computed, weighted by their responses; this is the SGD of the subimage. The SGD is similar to the DEO, except that (a) it takes into account the magnitude of contrast, and (b) it does not explicitly tokenize edges by non-maximum suppression. Voting on edge direction does, however, constitute a form of tokenization, so SGD is considered a sparse measurement function.

Edge strength

In addition to measuring the direction of the edges in the image, the strength of those edges can also be measured. The edge strength (ESTR) measure is computed as the average directional response over all filter directions (as computed for SGD). ESTR is simply an average 'edginess' for an image, and so is a dense measurement.

Edge density

A fourth edge-based measurement is the density of edges in the subimage (ZC). This gives a rough measure of the visual complexity of the scene. It can be computed by convolving the subimage with a Laplacian of Gaussian filter (using a 3×3 mask), and then counting the number of vertical and horizontal pixel boundaries which constitute zero-crossings of the convolved image. The fraction of such zero-crossing boundaries constitutes the edge density of the subimage. While fairly fine-grained, ZC relies on picking out edge fragments, and so is sparse.

Texturedness

Another simple measure is more specifically sensitive to image texturedness. A pixel's 'texturedness' is defined as the number of neighboring pixels whose intensities differ from it by more than a fraction $\vartheta_{\mathrm{diff}}^{\mathrm{TEX}}$ of the (sub)image's mean. The overall texturedness for a subimage is computed as the average of pixel texturedness on the subimage[1]. This measure will not be much confused by edges, as areas will predominate by the number of pixels involved in the average. Texturedness averages a local measure over the entire subimage, so is a dense operation.

Relative intensity

We can also roughly estimate the average reflectance of surfaces in a scene by measuring the relative intensity of a subimage compared to the entire image. This is done by computing the ratio of the intensity mean of each subimage to the intensity mean of the entire image. This normalizes for the ambient illumination level. Naturally, this measure will not be stable if the light source direction is changed such that the shadow structure of the scene changes. In fact, this is the least useful of the measurement functions tested. Relative intensity is clearly a dense measurement.

[1] This measurement function intentionally uses a simplistic notion of texturing, in order to demonstrate the power of the overall approach. More sophisticated texture measures could be used, perhaps based on the spatial-dependence matrices of [Haralick *et al.*, 1973].

Other measurements

In addition to the measurement functions described above, some others were tested and found wanting. They are briefly described here, along with why we think they failed.

Vertical line count This measure extracted vertical lines that extended the full height of the subimage and counted them. The intuition was to take advantage of the strong vertical orientation of our office environment. It didn't work too well, though, probably because vertical lines tend to come in dense patches so that small motions could change the count considerably.

Intensity centroid We also tried using measurements based on the grey-level centroid for something like region segmentation. Unfortunately, these measurements are very sensitive to camera motion for obvious reasons, which are difficult to correct for.

There is also a great deal of previous work in vision that can be applied to measurement function development. For example, texture analysis (as done by Witkin [1981]), color histograms [Swain, 1990], or shading (as in [Koenderink and van Doorn, 1989] or [Pentland, 1989]) might provide good physically-based measurement functions.

Evaluating measurement functions

What makes some measurement functions useful and others not? We would like a succinct characterization of measurement 'goodness' which could be used when designing measurement functions for a given environment. We can derive criteria for good measurement functions by considering the properties demanded by place recognition. First of all, views from a single place should look similar. We call this the *stability* criterion. We can measure the stability of an image measurement by looking at how much viewpoint motion it takes to find a non-matching image signature. The more motion it takes to 'break' a measurement, the more stable the measurement is. Roughly speaking, then, dominant edge direction is stable, since the features being measured change slowly and rarely disappear from view after a small movement (this argument is made more formally by Nelson [1989]). Another desirable

Name	Description
DEO	Dominant edge orientation
SGD	Significant gradient direction
ZC	Fraction of zero-crossing edges
TEX	Texturedness of subimages
ESTR	Average edge contrast
RI	Relative subimage intensity

Table 3.1: Summary of measurement functions used in this chapter.

sort of stability is invariance with respect to lighting conditions. Unfortunately, it is nearly impossible to achieve in general, so the robot may have to store different signatures for different lighting conditions. But if shadows are rare or small, they will not often affect results.

Another criterion, *distinguishability*, is that views from distant configurations should have different measurements, so that recognition will be reliable. More practically, this becomes a requirement that different physical layouts (geometry, reflectance, etc.) should look different under the measurement. Since the information in the image is being reduced severely, this criterion will not be completely satisfied; however, measurements that come close to describing inherent physical properties of the scene will be preferred. Thus, since contrast edges usually denote physical structures in the world (occlusions, surface markings, and so on), we expect dominant edge orientation to be a distinguishing measure.

3.2.2 Signature matching

Similarity metrics

We consider here signature matching methods which test to see if some 'similarity metric' is above a given threshold. If the robot wishes to test the hypothesis that it is at a particular place, an allowable signature match permits the hypothesis to be accepted. If competing hypotheses must be disambiguated, the match with highest similarity is chosen.

Nelson [1989] suggests using the fraction of identical elements in two signatures (which he calls 'patterns') as a similarity metric. He uses discrete measurements—for real-valued measurements we can define two values to be 'identical' if their difference is less than a threshold. So, if s and t are $n \times n$ image signatures, we have the *pattern similarity* metric:

$$\mathbf{sim}_{\mathrm{pat}}(s,t) = \frac{1}{n^2} \sum_{ij} \mathrm{id}(s_{ij}, t_{ij})$$

where

$$\mathrm{id}(v_1, v_2) = \begin{cases} 1 & \text{if } \mid v_1 - v_2 \mid \leq \vartheta_{\mathrm{id}} \\ 0 & \text{otherwise} \end{cases}$$

This metric corresponds to using a 0–1 loss for measurement estimation [Berger, 1985], and is reasonable if measurement deviations are usually less than ϑ_{id}, but occasionally are completely unpredictable. This will typically be the case if the measurements are based on tokens representing features in the world which may suddenly disappear from view when the viewpoint changes. If features are sparsely distributed, few will move out of a given image region under small motions—allowing pattern similarity to match the derived signatures. A match for signatures s and t, with measurement function M, is *permitted* under pattern similarity if $\mathbf{sim}_{\mathrm{pat}}(s,t) \geq \vartheta^{M}_{\mathrm{pat}}$.

A second similarity metric is based on the *root-mean-square-difference*, or *cross-correlation*, of the signature array elements. To wit:

$$\mathbf{sim}_{\mathrm{rms}}(s,t) = -\sqrt{\frac{1}{n^2} \sum_{ij} (s_{ij} - t_{ij})^2}$$

This similarity metric corresponds to using a squared-difference loss for measurement estimation [Berger, 1985], or a Gaussian probability density on measurement deviation[2]. More generally, this metric is reasonable whenever the chance of a particular measurement deviation decreases smoothly with deviation's magnitude. In particular, dense measurement functions vary continuously with robot motion, so $\mathbf{sim}_{\mathrm{rms}}$ will be appropriate. A match is permitted (as above) if $\mathbf{sim}_{\mathrm{rms}}(s,t) \geq \vartheta^{M}_{\mathrm{rms}}$.

[2]It is possible that matching could improved by reducing the influence of outliers, using robust estimation techniques [Hampel *et al.*, 1986]. This kind of approach has been applied to good effect for pixel-level matching, see [Black, 1992] for example. We have not yet examined the effects of robustification on signature matching.

Offset and cross-scale matching

Recall that one of our goals for matching is stability—signatures from nearby views should be 'close'. However, even with reasonably stable measurement functions, signatures will only match by the above criteria if they line up almost exactly. For example, with 5×5 signatures and a 25 degree field of view, a rotation of only 5 degrees will induce a signature match where no matching parts of the scene are compared. But if matching is done at a horizontal offset of one column, the signatures will match. Thus we see that matching signatures at horizontal offsets can correct for rotation about the vertical axis. Of course, under planar projection, the corresponding subimages will not be identical, but there is empirical evidence (see Section 3.4.1) that the difference is tolerable.

Translation can be corrected for similarly. Translation perpendicular to the optical axis results in simple image translation and can be dealt with by offset matching as above. If translation is purely along the optical axis, then the focus-of-expansion (FOE) is at the center of the image. Provided that objects in the scene are reasonably far from the observer and motion is small, the image change can be approximately modeled as pure expansion. If this is the case, then the center portion of a high-resolution signature at the original position should match a lower-resolution signature at the forward position. Other translations can be approximately corrected for by combining multiscale and offset matching.

3.2.3 Using multiple measures

Despite our best efforts, any single measurement function that is designed will be far from perfect, due to the amount of information reduction inherent in the process. Hence we must use additional cues to filter allowable matches and improve recognition. The most obvious is to use several measurement functions together for matching. The basic idea is quite simple. Several signature databases are kept, one for each of the different measurement functions used. A suggested image match is permitted if the several signature matches implied by the hypothesis are themselves permitted. Even though one particular measurement function may accidentally allow an invalid match, it is less likely that several will. This is only true if the measurement functions are, in some sense, independent. The notion of measurement independence is discussed in more detail later.

More precisely, matching using multiple measurement functions is done as follows. Given a match hypothesis, corresponding signatures are checked to see if a match is allowed, based on the measurements' thresholded similarity metrics. Note that different measurements may use entirely different match criteria. If fewer than a given number of the signature matches are disallowed, then the match is allowed. If enough measurement functions are used, then allowing one or two mismatches avoids some false negatives without unduly hindering positive recognition.

Choosing the 'best' match among several permissible alternatives is more difficult, however, since signature similarities for different measurements are not generally commensurable. The strategy we currently use is to designate one measurement as 'special' and rank matches based on similarity with respect to that measurement. There are a host of other methods for producing orders using all an image's signatures, but this simple method seems to work reasonably well (for a good choice of special measurement).

Dependence

For a set of measurement functions to be usefully combined, they should be 'independent' in some sense. If different measurement functions measure different image properties, then they will tend to disagree on accidental matches of different scenes, and hence the set will perform better than its elements.

A new measurement function will generally improve performance if it allows the system to rule out enough more incorrect matches than before (assuming reasonable stability). A simple way to evaluate this sort of independence is to estimate the probability that the measurements agree on an incorrect match. What we would really like to determine is the proper tradeoff between having more measurement functions, which cost us both in computation and in good matches improperly rejected, and having fewer measurement functions, which cost us in bad matches improperly accepted. However, these costs depend on the place of signature matching in the overall robot system, as well as on the robot's goals and behavior patterns. Hence a full utility analysis would seem to be impossible, so we must be satisfied with a raw measure of dependence, derived as follows.

If we let I_1 and I_2 denote random images, m_i denote the event that I_1 and I_2 are deemed to match under measurement function i, and $I_1 \sim I_2$ denote the event that the images arise from the same scene, we can write the *dependence* of a set of measurement functions, D, as:

$$D = P\left(\bigwedge_i m_i \middle| (\bigvee_i m_i) \wedge (I_1 \not\sim I_2)\right)$$

D denotes the probability that all the measurement functions allow an incorrect match, given that one of them does. If D is high, then most of the measurement functions are not doing much, most of the time. We can evaluate D empirically by first simplifying:

$$\begin{aligned} D &= P\left(\bigwedge_i m_i \mid (\bigvee_i m_i) \wedge I_1 \not\sim I_2\right) \\ &= \frac{P\left((\bigwedge_i m_i) \wedge (\bigvee_i m_i) | I_1 \not\sim I_2\right)}{P(\bigvee_i m_i | I_1 \not\sim I_2)} \\ &= \frac{P(\bigwedge_i m_i | I_1 \not\sim I_2)}{P(\bigvee_i m_i | I_1 \not\sim I_2)} \end{aligned}$$

This value can be estimated from a representative sample of images S (with correct matches known) after computing the sets of incorrect matches for each measurement function i, $B_i \subset S \times S$, by

$$\frac{|\bigcap_i B_i|}{|\bigcup_i B_i|}$$

This provides a direct way to approximate the dependence of a set of measurement functions. Section 3.4.1 evaluates a set of measurement functions by this criterion, and shows the correlation between independence and improvement in recognition. The difficulty with this method lies in obtaining a representative set of images. This difficulty is, however, inherent to the problem, as any method of evaluating image-based recognition must rely on the notion of a 'representative sample'. If, as we propose, the system is to be used on a mobile robot, then perforce the images gathered by the robot will be representative. The validity of using smaller samples can be checked empirically by comparing the results with those given by using larger samples, and a good sample size (for an environment class) can be chosen.

3.2.4 Hypothesis generation

The above discussion on matching image signatures assumes that all signatures in the database are checked as possible matches. Since in practice the robot may need to store thousands of images, it must be able to find the relevant ones effectively. Such hypothesized matches can be found relatively easily if the robot has strong expectations about its current location. If the robot believes it is at one of a small set of places in its map, then it need only search signatures corresponding to those places to disambiguate its position. If it fails to rule out all but one possibility, then it can look around some more and repeat the process with another image.

If, on the other hand, none of the expected signatures match, or the robot has no expectations at all, the robot must figure out where it is from scratch. If it has knowledge about its geometric position from odometry, that may be used to generate candidate places the robot might be. If not, or if its positional information is not good enough, the robot must solve the *kidnapped robot problem.* It is as if the robot fell asleep and then was silently kidnapped and moved to some new location by gremlins. When the robot wakes up, it must come up with some idea of where it is based on purely perceptual cues. (We ignore here the use of experimentation.) Thus, we require an efficient way to index a signature database for effective retrieval of reasonable match hypotheses.

The solution is to index signatures using k-d trees (a k-dimensional generalization of binary search trees [Preparata and Shamos, 1988]). Direct search can be efficiently performed on a k-d tree for points falling within intervals of $\mathbb{R}^k$. A more flexible search strategy for our purposes, though, is a variety of *spiral search* [McDermott, 1981b]. Spiral search uses a heuristic strategy to find points in the tree progressively further and further from a given starting point. Note that k-d tree search can be efficiently implemented on massively parallel machines using marker-passing.

For our application, the simplest thing to do would be to treat $n \times n$ signatures as points in $\mathbb{R}^{n^2}$ and index them in the tree directly. However, this does not take into account the element-wise flexibility of our similarity measures (some elements may be very different) or offset/multi-scale matching. These problems can both be ameliorated by indexing on signature subarrays. If we index each signature under each of its columns, offset match hypotheses can be generated using markers. Each signature keeps a vector of markers, one for each possible offset. Every time a column of the input signature is 'matched' to an index column, the corresponding offset marker is incremented; when it goes above some fraction of n, a match hypothesis at that offset is proposed. This also eases the problem of some elements being far off, since only some columns must be nearby. This approach may be generalized by indexing based on arbitrary sub-signatures—supporting generation of cross-scale match hypotheses. However, there are an exponential number of such possible indices, so only 'reasonable' ones should be used. A good way to filter indices is to only use those which are relatively unambiguous; if an index points to hundreds of signatures it doesn't constrain the hypothesis space much.

3.3 The Recognizability Problem

There are several difficulties with the naive approach to place recognition described above. First, if all images seen are stored, the signature database will grow enormously. Thus, some method is needed to choose which signatures are most useful for recognition and should be stored. This brings us to the fact that a large fraction of the views in many environments are ambiguous, and say little or nothing about the viewer's location. Consider how many viewpoints in an office building give an image of a blank wall. In the recognition experiments described in Section 3.4, more than 35% of images were found to be ambiguous. Measuring ambiguity by seeing how many signatures in a database are matched by an input image is expensive, in both time and space (since ambiguous signatures must be stored for testing). Hence, finding distinct images should not depend on direct examination of a signature database. A final consideration is the problem of *stability*. The difficulty here is that there are configurations where a change in viewpoint can radically change the view seen. Thus, if the naive approach is applied at an unstable viewpoint, many more views will need to be stored for a small region, to cover all the possible views. The question addressed in this section is how to choose viewpoints which are distinctive and stable. Such viewpoints (and the images they induce) are termed *recognizable*.

To find recognizable viewpoints, we use the fact that place recognition is performed from a mobile platform, and hence the camera can be moved during recognition. Most work in vision that uses camera motion can be divided into two categories:

active vision and *sensor planning*. Active vision techniques focus on temporal integration of visual information over a controlled trajectory to improve interpretation (see [Ballard, 1989]). Usually, low-level intensity/feature models are used to determine camera motion (for, eg, fixation as in [Ballard and Ozcandarli, 1988]). Visual servoing of some sort is often used for controlling camera motion, though arbitrary motion can also be helpful [Nelson and Aloimonos, 1988]. Sensor planning, on the other hand, addresses the problem of finding good viewpoints for object recognition and registration [Tarabanis *et al.*, 1990]. This is done by optimizing some criterion of image goodness, using measures such as number of visible useful features or expected accuracy of image measurements. A high-level model is used to determine the best viewpoint subject to kinematic constraints, given some previous estimate(s) of the target's pose. The problem can also be formulated as a decision-theoretic problem of choosing the viewpoint with highest expected utility [Hager, 1990]. This can be iterated to obtain successively better estimates (of target identity or pose).

The problem of finding recognizable viewpoints is similar to the classical sensor planning problem, in that we seek to optimize some measure of viewpoint 'goodness'. However, we do not have an accurate and complete geometrical model of the target, and so such approaches cannot be used. The approach taken here uses a coarse sort of visual servoing to find distinctive viewpoints. This is done by locally maximizing 'signature distinctiveness' to home in on recognizable images. This has an advantage that it does not require explicit modeling of environment structure, only a heuristic notion of signature 'distinctiveness'.

3.3.1 Distinctiveness

Generally speaking, for recognition we should prefer signatures which rarely match others, since a signature will always match itself. Given a representative signature database, then, a signature's distinctiveness can be directly measured, by seeing how many signatures in the database match it (this is how ambiguous images were filtered out for the recognition experiments described below). For batched experimentation this is satisfactory; however it becomes impractical when (quasi-) real time performance is needed (as on an autonomous robot). Hence we need *distinctiveness metrics* that can be computed directly from an image signature; a *distinctiveness value* can then be used to classify images as distinctive or ambiguous. This seems a plausible goal since we, as humans, can usually easily tell whether or not an image is distinctive. Given a distinctiveness metric, the locally 'best' view is chosen by searching for a local maximum of the metric. This idea is similar to that used by Kuipers and Byun [1988] for place definition, though our purposes are different. Here, we deal only with rotational search, but the approach can be extended to include translation as well.

While the main motivation for using locally distinctive signatures is to improve recognition by filtering out ambiguous views, there are other benefits as well. Most obvious is the reduction in the size of the necessary signature database. Assuming that distinctive images are relatively rare, substantially fewer signatures will be stored for each place, since only the most distinctive views will be recorded. A further benefit that we would hope to obtain would be greater recognition stability, since active search should compensate for differences in original viewpoint. Such stability is not a priori obvious though, and must be demonstrated. This is done here empirically, as described below in Section 3.4.2.

3.3.2 Distinctiveness metrics

Due to the unpredictable complexity of the world, we designed distinctiveness metrics heuristically and evaluated them empirically. The basic idea is to ask what sorts of views tend to be ambiguous, and what signature features predict those views. We consider ambiguous views since they seem to be a more homogeneous class than distinctive views. Naturally, different measurement functions will require (in general) different distinctiveness metrics. Also, since views considered ambiguous in one environment may not be in another (consider a view of a tree in a forest, or in an office building), different environments may require different distinctiveness metrics. Discussion of the issues of operation in multiple environments is beyond the scope of this chapter, however. The environment which we consider is the usual indoor office environment. This means that the main class of ambiguous images will be blank walls. Below we discuss distinctiveness metrics developed for the measurement functions used here (see Section 3.2.1 above). Since relative intensity gave indifferent recognition results, it was omitted from consideration in this part of the work.

For the indoor environment, we examined a number of possible distinctiveness metrics, only some of which worked. The simplest way to measure the featurelessness of a signature is to estimate its homogeneity[3]. The equation of homogeneity with ambiguity is a contingent one, dependent on our choice of indoor environments. Different criteria may be needed to diagnose ambiguity in different environments.

For real-valued measurements such as TEX and ZC (refer to Table 3.1), the homogeneity of a signature can be taken as inversely proportional to the variance of the signature array values, $\frac{1}{n^2}\sum_{i,j<n}(v_{ij} - \bar{v})^2$ (call this distinctiveness metric SigVar). Since TEX itself gives a measure of image heterogeneity, the average value of signature elements (MeanVal) was also found to correlate well with ambiguity.

SigVar is not applicable to DEO, however, since its values are angles (and so variance is not well-defined). A similar idea can be applied, however, by calculating the estimated entropy of the signature values, taking them to come from a single probability distribution. It must be emphasized, though, that this is only used to measure signature homogeneity—if all values are the same, entropy is minimized; if they are equally distributed, entropy is maximized. If the possible values are indexed by i, and the fraction of signature elements with values i is p_i, the signature entropy (SigEnt) is computed as $\sum_i p_i \log p_i$.

As it turns out, none of the metrics above works well for SGD. SigVar is again inapplicable due to SGD being angle-valued, and SigEnt has trouble, it seems, because lighting can cause noticeable contrast variations over large areas of homogeneous regions. However, a metric that does give satisfactory results for SGD is the fraction of array elements for which a significant gradient was registered. When there is any significant environmental structure, a significant intensity gradient will usually be generated; empirically, the converse seems to be true as well. Hence, the noticeable gradient metric (NoGrad) will usually be low only for ambiguous featureless views.

[3]Note that signatures, not image intensities, are homogeneous. For example, wallpaper with a repetitive but highly textured pattern would be judged to be homogeneous, since it is self similar on a large scale, despite being visually interesting.

3.3.3 Search

Since it is impractical to examine all viewpoints in a neighborhood for a distinctive image, some sort of search is required. In this chapter, we only consider rotational search, so a single search granularity, $\epsilon_V{}^\circ$, must be specified, such that a view orientation within $\epsilon_V{}^\circ$ of a true distinctiveness maximum is acceptable. We mainly consider here two search algorithms: a linear hillclimbing method and an interval subdivision method. The hillclimbing method works by sampling images every $\epsilon_V{}^\circ$ (always turning clockwise, say), until a three-image window is found whose middle image is a distinctiveness maximum. If the expected distance to a maximum is $\bar{V}^\circ$, this algorithm will examine $\mathrm{O}(\frac{\bar{V}}{\epsilon_V})$ images.

For interval subdivision, we used a form of bisection line search, since no derivative information is available. Also, rather than assume that a maximum has been bracketed, we search for the local maximum within a predetermined interval. The algorithm is given a starting search size $V_0{}^\circ$. The starting interval is then that of size $V_0{}^\circ$ clockwise (say) of the starting orientation. The current interval is iteratively trisected[4], and the side containing the most distinctive signature is chosen as the next interval. This continues until the search interval is smaller than $\epsilon_V{}^\circ$. The trisection algorithm processes 2 images on each trisection, and performs $\mathrm{O}(\log(\frac{V_0}{\epsilon_V}))$ trisections. Clearly, if $\bar{V}$ is known, choosing $V_0 = \bar{V}$ makes trisection more efficient than hillclimbing, but this is hard to do in general. However, in practice, trisection was more efficient than hillclimbing, even with only a rough guess at a good V_0.

3.3.4 Using multiple measurements

Using multiple measurement functions for conjunctive filtering can dramatically improve recognition. However, finding distinctive images based on multiple measurement types is nontrivial, since distinctiveness is not comparable across measurement types. Rather than try to guess at a combination function (for which there is no principled method), we can make use of the fact that we are trying to maximize distinctiveness. This means that we only need a comparison relation giving a partial

[4]Golden section search could be similarly used with a further efficiency improvement; trisection search was used due to ease of implementation. Also, some results seem to show that trisection is more stable, probably since more data are used.

order (preferably nearly total). To compare two signature sets S_1 and S_2, we can count up the number of measurements for which each corresponding signature in S_1 has higher distinctiveness than that in S_2 and vice versa. The set with a higher count is preferred. This is extended for trisection search by also counting the number of distinctiveness preferences of each internal view to the interval endpoints. In general, preference comparison leads to the same results as distinctiveness comparison, though there are cases in which preferences cannot distinguish between possibilities. Note also that the preferred viewpoint may not be distinctive with respect to all measurements; a higher preference is still indicative of a better viewpoint.

3.4 Experimental Results

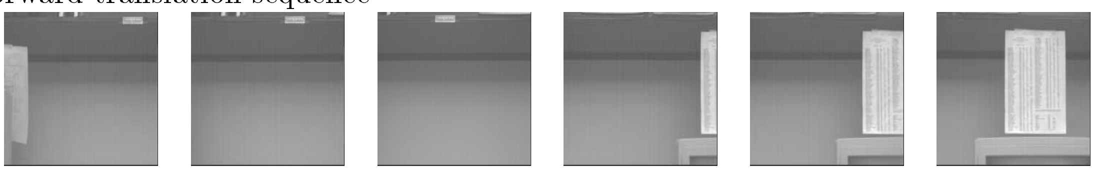
3.4.1 Place recognition

We conducted experiments on image matching using the measurement functions described above in Section 3.2.1. Images were taken by a CCD camera with a 25mm lens, giving slightly more than 12 degrees of view. The images were reduced by pixel averaging from 640×480 to 60×45 before signatures were computed. The bulk of the experiments described here were performed on a corpus of 168 images taken from 8 different locations in our building. At each place a sequence of 21 images was taken, with a rotation of around 6 degrees between adjacent images. Thus a total of about 130 degrees were covered at each place. Note that adjacent images overlap by about half and non-adjacent images barely, if at all. This gives us an easy way to automatically check ground truth for matching.

Except as otherwise noted, system parameters were as follows. Signatures were 8×8 grids. Mismatches were generally disallowed. Matching thresholds were: $\vartheta^{\mathrm{DEO}}_{\mathrm{pat}} = 0.6$, $\vartheta^{\mathrm{SGD}}_{\mathrm{pat}} = 0.6$, $\vartheta^{\mathrm{ZC}}_{\mathrm{pat}} = 0.5$, $\vartheta^{\mathrm{ESTR}}_{\mathrm{rms}} = 25$, $\vartheta^{\mathrm{TEX}}_{\mathrm{rms}} = 0.5$, and $\vartheta^{\mathrm{RI}}_{\mathrm{rms}} = 0.1$. Measurement parameters were: $\vartheta^{\mathrm{DEO}}_{\mathrm{grad}} = 70$, $\vartheta^{\mathrm{SGD}}_{\mathrm{der}} = 50$, and $\vartheta^{\mathrm{TEX}}_{\mathrm{diff}} = 0.05$. Ambiguous images were taken to be those matching more than 5% of the database.

Rotation sequence

Forward translation sequence



Transverse translation sequence

Figure 3.3: Image sequences for measurement stability evaluation (every other image).

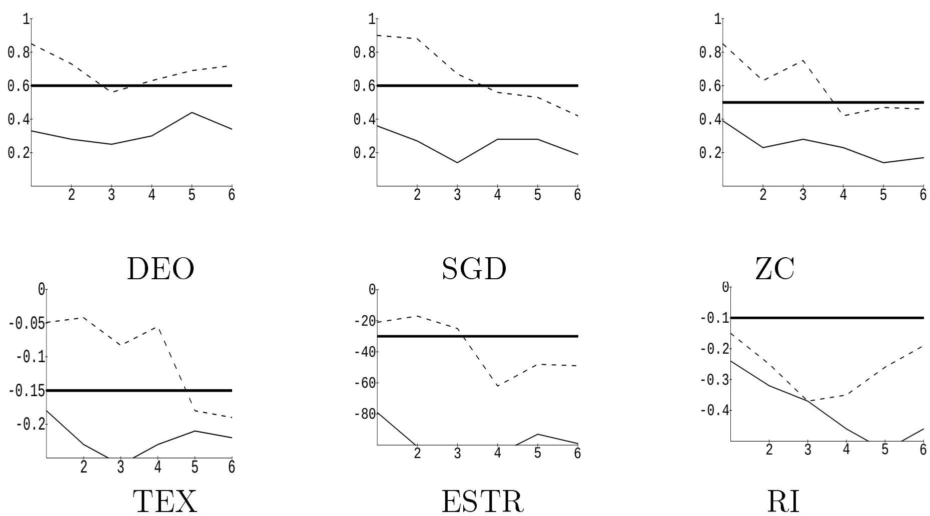


Figure 3.4: Rotation sequence stability plots. Signature similarity is plotted versus angular distance (in units of 2°).

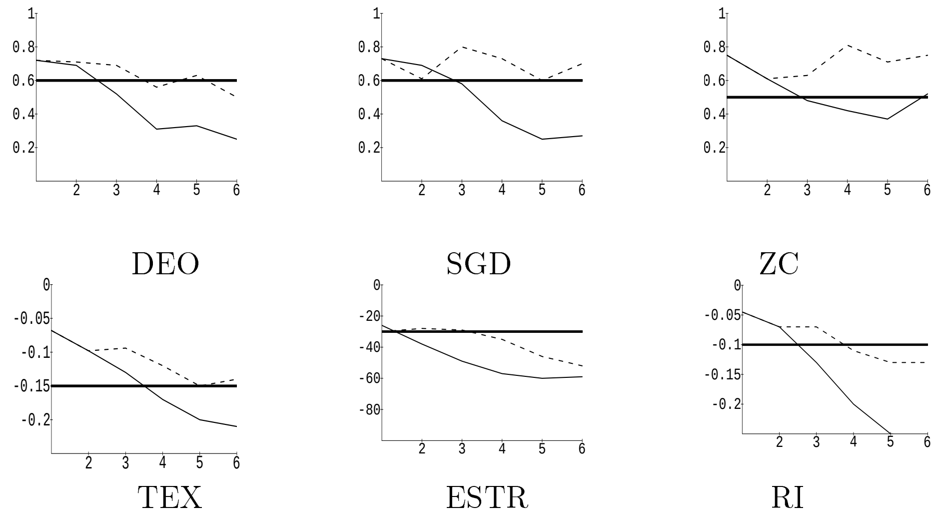


Figure 3.5: Forward sequence stability plots. Signature similarity is plotted versus distance (in units of 5cm).

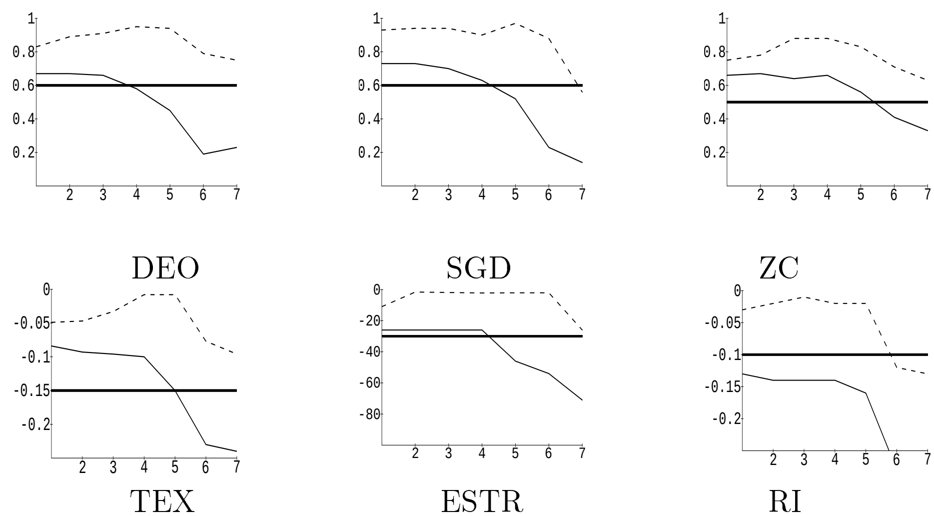


Figure 3.6: Transverse sequence stability plots. Signature similarity is plotted versus distance (in units of 5cm).

Measurement evaluation

Stability

We first evaluated the stability of individual measurements. Stability was estimated separately for rotation and translation. Rotational stability was evaluated by plotting the match similarity change with rotation at intervals of about $2°$. Translation was evaluated for forward and transverse movement at intervals of about 5cm. The images are shown in Figure 3.3. Both full signature similarity and best offset similarity were calculated between the first image and all other images in each series. These similarities are plotted against image number in Figures 3.4, 3.5, and 3.6. Solid lines represent matching without offset, dashed lines matching at best offset, and thick horizontal lines show the matching threshold used.

Most of the plots show a reasonable degree of stability for non-offset matching; the exceptions are the rotation matches, which is understandable since a 2 degree rotation is an offset of just over one grid cell[5] (at 8×8). The usefulness of offset matching is immediately apparent; most measures allow offset matching to a distance of about 25cm forward, 35cm sideways, and 9 degrees of rotation. The main exception is relative intensity, which doesn't match at all for any rotation. It appear that this is due to the greatly changing global illumination in that sequence; it confuses RI's normalization. In our experience, these plots seems representative of the measurements' behavior. The stability region implied, 25cm$\times$35cm, is probably smaller that strictly necessary by the fact that our camera has a fairly narrow field of view.

Distinguishability

The distinguishability of measurement functions was evaluated, based on our image corpus, by computing, for each measurement function, the average number of matches found per image for which a match is found. Since there are two correct matches for

[5]The use of a 2 degree rotation increment was due to limitations in our equipment. The possibility exists that the results for offset matching under rotation are due to aliasing. That is, however, unlikely, since the rotation increment was not a whole number of grid cells and results are consistent over all measurement functions. More detailed experiments should definitely be conducted to test this possibility.

Measurement(s)	Average number of matches
DEO	1.66
SGD	2.26
ZC	3.43
TEX	3.39
ESTR	1.68
RI	2.74
DEO,SGD	2.08
DEO,TEX	2.19
TEX,ESTR	1.59
ZC,RI	2.14
DEO,SGD,TEX,ESTR,ZC	1.74

Table 3.2: Estimated measurement function distinguishability.

nearly every image in the corpus (its predecessor and successor), we should expect this average to be something less than 2. The greater the number, the less distinguishable the measurement. To avoid being confounded by inherently ambiguous images, this average was computed while filtering ambiguities out. Table 3.2 summarizes the results; we also evaluated some sets of measurement functions working in tandem. The most distinguishable measurements seem to be DEO and ESTR, and the least ZC and TEX. As expected, distinguishability increases when measurement functions are combined.

Independence

Recall that in Section 3.2.3 we discussed a way of evaluating the dependence of sets of measurement functions in the context of signature matching. Since efficiency dictates that as few measurements should be used as possible, effort should be taken to choose a reasonably independent set. We evaluated pairs of measurement functions described above in terms of the approximate dependence formula derived above. In this case, the numerator counts the number of non-adjacent image pairs in our sample which matched according to both measurements being evaluated, and the denominator those

	SGD	ESTR	ZC	TEX	RI
DEO	0.08	0.10	0.05	0.06	0.17
SGD	–	0.17	0.05	0.06	0.10
ESTR	0.17	–	0.10	0.08	0.22
ZC	0.05	0.10	–	0.26	0.09
TEX	0.06	0.08	0.26	–	0.05
RI	0.10	0.22	0.09	0.05	–

Table 3.3: Matching dependencies for pairs of measurement functions.

which matched one of the two. Table 3.3 shows the results of applying this dependency metric to pairs of measurement functions on the image corpus described above.

Some of the results in the table immediately make sense. For example, the high degree of dependence between TEX and ZC—since TEX measures texture by large intensity variations between neighbors, it is to be expected that those same pixels would tend to abut Laplacian zero-crossings. Some results are somewhat surprising, though. The high degree of dependence between RI and ESTR seems strange at first; but it is probably due to the fact that image regions with high edge strength will have large intensity variation, and hence the average intensity will tend to not be extreme. This correlates the two measures. A similar argument explains the lesser, though significant, dependence of RI with DEO and SGD. The result that is perhaps most surprising is the lack of dependence between DEO and SGD. Since these were both intended, in a way, to measure 'edge direction', one would expect their matches to be highly correlated. However, since SGD does not extract edge tokens, but merely looks at high-magnitude gradients, it also picks up curved and specular surfaces, as well as surfaces near a light source. These results support our dependence measure.

Recognition

Finally, we tested the recognition capabilities of the system on our image corpus. The images were each processed to produce a set of six 8×8 signatures, one for each combination of image and measurement function. The following experimental procedure was performed twice, once as is, and again filtering out ambiguous images

Measures	M	BC	4C
DEO	41 (24%)	20 (49%)	24 (59%)
SGD	39 (23%)	26 (67%)	30 (77%)
ZC	14 (8%)	08 (57%)	09 (64%)
TEX	23 (14%)	08 (35%)	21 (91%)
ESTR	25 (15%)	21 (84%)	24 (96%)
RI	38 (23%)	17 (45%)	24 (63%)
DEO, SGD	51 (30%)	39 (76%)	46 (90%)
DEO, TEX	63 (38%)	39 (62%)	51 (81%)
TEX, ESTR	29 (17%)	26 (90%)	28 (97%)
ZC, RI	44 (26%)	25 (57%)	29 (66%)
DEO, SGD, TEX, ESTR, ZC	54 (32%)	43 (80%)	50 (93%)
DEO, SGD, TEX, ESTR, ZC, RI	50 (30%)	37 (74%)	45 (90%)

Table 3.4: Recognition statistics with ambiguity filtering. Ambiguous images were neither checked for matches nor matched against. M = images for which matches were found, BC = preferred matches which were correct, and 4C = correct match in the top 4. Percentages shown in parentheses are as follows: images matched vs. images checked, best match correct vs. images matched, correct match in top 4 vs. images matched.

(those matching more than 5% of the corpus). For different combinations of measurement functions, each image's signatures were matched at all offsets, up to 3/4 of the signature, against all other images' signatures. Several statistics were accumulated:

- The number of ambiguous images found (presented above),
- The number of images for which a match was found,
- The average number of matches per image with one (presented above),
- The number of images for which the highest-ranked match was correct,
- The number of images for which a correct match was among the 4 highest ranked. This gives us an idea of how well we could do with geometric and contextual filtering.

Measures	M	BC	4C
DEO	158 (94%)	18 (11%)	58 (39%)
SGD	166 (99%)	53 (32%)	66 (40%)
ZC	168 (100%)	3 (2%)	9 (5%)
TEX	168 (100%)	13 (8%)	24 (14%)
ESTR	163 (97%)	24 (15%)	30 (18%)
RI	165 (98%)	13 (8%)	27 (16%)
DEO,SGD	139 (83%)	49 (35%)	62 (45%)
DEO,TEX	148 (88%)	39 (26%)	39 (26%)
TEX,ESTR	163 (97%)	28 (17%)	33 (20%)
ZC,RI	159 (95%)	26 (16%)	34 (21%)
DEO,SGD,TEX,ESTR,ZC	128 (76%)	47 (37%)	61 (48%)
DEO,SGD,TEX,ESTR,ZC,RI	116 (69%)	37 (32%)	49 (42%)

Table 3.5: Recognition statistics without ambiguity filtering. The arrangement of this table is the same as that in Table 3.4.

The results for various combinations of measurement functions are summarized in Tables 3.4 and 3.5.

The first thing to note is that we can get the excellent accuracy of around 90% when filtering ambiguities. Also, most of the time adding more measurement functions increases accuracy. In those few places where percentage accuracy decreases, this is because fewer images are being classified as ambiguous, and the raw number of correct matches does increase. The relatively poor results without ambiguity filtering are mainly due to the high degree of inherent ambiguity of some of our images—the average number of matches found for each image was typically over 40 without ambiguity filtering. Also note that distinguishability seems to correlate with performance without ambiguity filtering, as expected. For example, ZC is the least distinguishable, and it gives the worst results. On the other hand, DEO and ESTR, which are much more distinguishable, give noticeably better results. Thus, these empirical results also support our evaluation metrics.

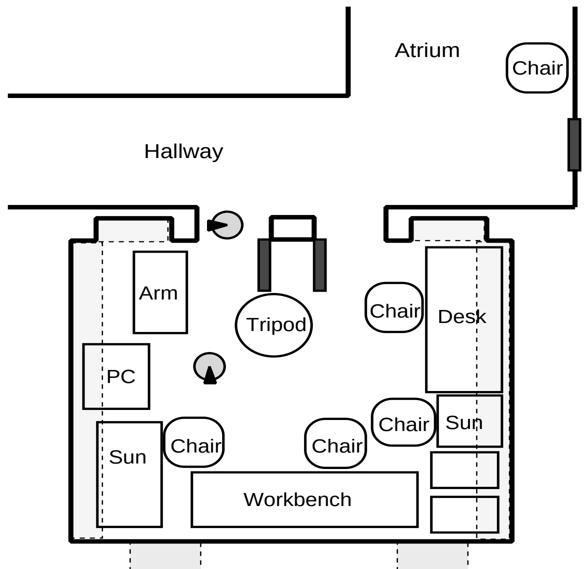


Figure 3.7: Approximate floor-plan of our laboratory. The two camera positions ("Door" and "InLab") used for rotational experiments are shown as grey circles, with the zero points for the static stability experiments (Figure 3.8) shown as black wedges.

3.4.2 Distinctiveness search

We performed a number of experiments to evaluate the performance of the active recognition methods described in Section 3.3 above. Experiments were performed using a CCD camera at a height of 110cm. As above, images were reduced by pixel averaging from 640×480 to 60×45 before signatures were computed. Some of the experiments below use a corpus of 291 images (a superset of that used in Section 3.4) taken at 11 different positions in the building using a 25mm lens. Those experiments not involving the image corpus used an 8mm lens with a larger field of view. At each position a set of images was taken with the camera at different orientations. Other experiments were performed using a pan-tilt platform in our laboratory. A drawing of the laboratory layout during the experiments is shown in Figure 3.7. The robot arm in the upper left and the electronics workbench were the most visually interesting things in the room.

Due to the difficulty of performing exhaustive and systematic tests using a mobile platform, two sets of experiments were performed to evaluate the signature distinctiveness method. The first are static evaluations, using the image corpus and rotational image sequences to evaluate ambiguity prediction and density of distinctiveness maxima. A second series of experiments was performed using a pan-tilt head to perform rotational distinctiveness maximization, and the results were evaluated. The details of the experiments and analysis are described below.

Distinctiveness and ambiguity

We evaluated the use of distinctiveness metrics for predicting ambiguity using the image corpus. For each measurement function, signatures at several resolutions were computed from each image in the corpus. For each signature, the program then found all other signatures that matched it, and computed the signature's distinctiveness using the appropriate metric. This then gave us a pair of numbers for each image i, a number of matching images M_i and a calculated 'distinctiveness' D_i.

The task is then to evaluate the hypothesis that local distinctiveness maxima are probably also unambiguous. The data for 8×8 signatures are plotted in Figure 3.8. From the shape of the plots it is clear that there is a strong relationship between the number of images matched and distinctiveness values. However, to properly evaluate the results, a more precise evaluation of the data is required. Due to the impossibility of modeling, even approximately, the underlying environmental structure that gave rise to the observations, we approach this problem indirectly. A threshold was first established on the number of images matched for calling an image ambiguous or not; this threshold was 5% of the database (14.6 images). We can then think of the distinctiveness values for 'ambiguous' images ($M_i \geq 15$) and those for 'unambiguous' images ($M_i \leq 14$) as two random variables, call them A and U. We then compare the two distributions to evaluate the reasonableness of our hypothesis. This was done using four different tests. The first two provide an intuitive feel for the overall properties of the data, and the latter two statistically test the hypothesis that distinctiveness values are unrelated to ambiguity.

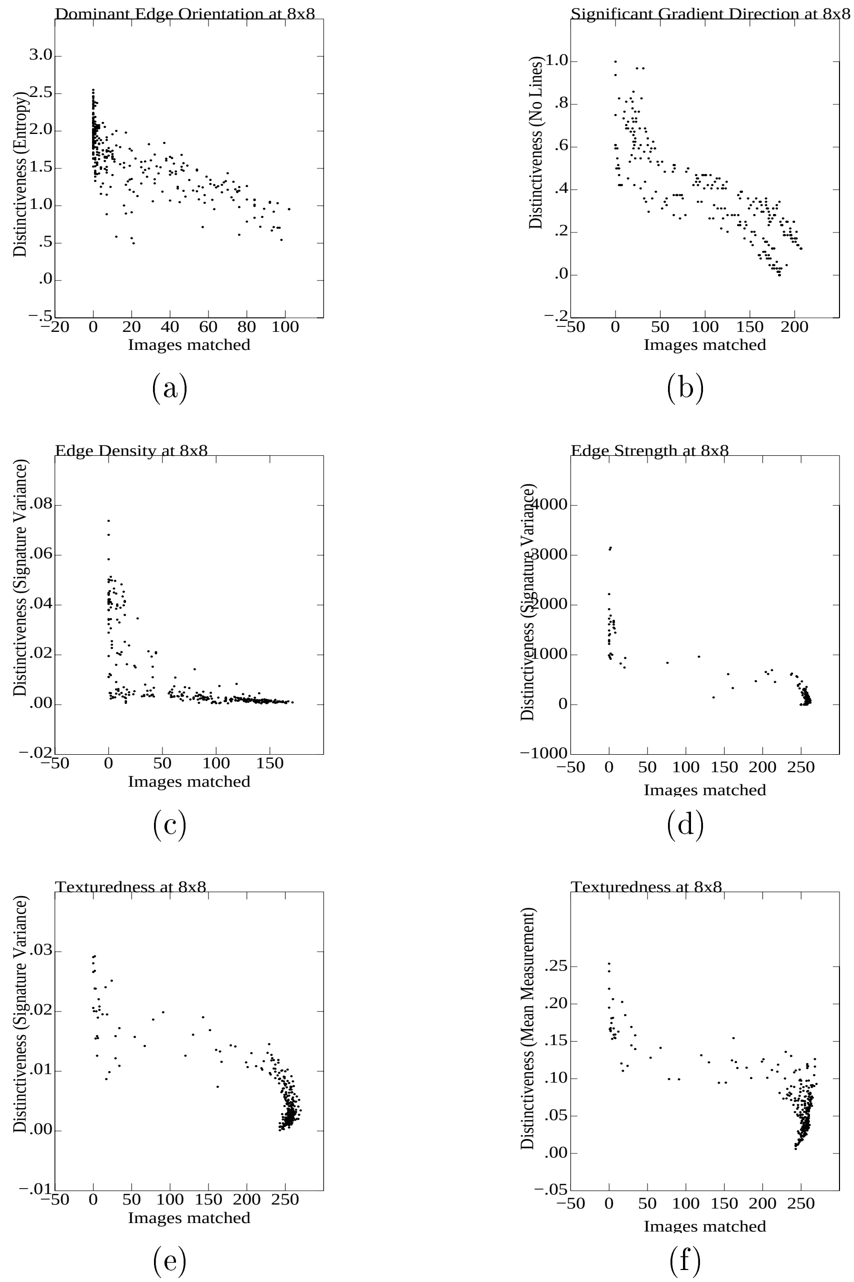


Figure 3.8: Representative plots of distinctiveness (y-axis) versus number of database images matched (x-axis), using 8×8 signatures. (a) DEO signatures using SigEnt. (b) SGD signatures using NoGrad. (c) ZC signatures using SigVar. (d) ESTR signatures using SigVar (e) TEX signatures using SigVar. (f) TEX signatures using MeanVal; note the similarity to (e).

- We calculated the ratio of the sample means $\bar{U}/\bar{A}$; if it is far from 1, then on average unambiguous images have distinctiveness noticeably different from ambiguous images.

- We estimated the probability of a random unambiguous image having greater distinctiveness than a random ambiguous image, $Pr(D_i > D_j \mid M_i < \vartheta \wedge M_j > \vartheta)$. This is evaluated as the fraction of image pairs with one unambiguous and the other ambiguous satisfying the former condition. If this probability is high (say, >90%), a strong argument can be made that unambiguous images have significantly greater distinctiveness than ambiguous ones.

- We tested the hypothesis that A and U come from distributions with the same median (which would be expected if distinctiveness was unrelated to ambiguity). Since there is no good statistical model for the data, we used the Mann-Whitney median U non-parametric test (described in [Fryer, 1966]). It uses rank sums to test the hypothesis that two samples of different sizes come from distributions with the same median. The procedure provides a z score that can be checked for significance against tables of the normal distribution. If the hypothesis of equal medians can be rejected at a reasonable level of significance (say, 1%), then we can conclude that a distinctiveness metric operates differently on ambiguous and unambiguous images.

- Finally, we used the Kruskal-Wallis H test to test the hypothesis that the two samples of distinctiveness values come from the same population. The H test tests whether sample rank sums are equidistributed as would be expected if the samples came from the same population. The procedure provides a number, H, which is used to perform a χ^2 significance test with 1 degree of freedom (since we have two samples). Again, if the hypothesis (of a single population) can be rejected with reasonable significance, we can say that distinctiveness and ambiguity are related.

While no one of these tests is sufficient to establish our main hypothesis, if a distinctiveness metric satisfies all of them, the evidence is strongly suggestive that distinctiveness maxima are generally unambiguous. The results of these tests for the various measurement functions/distinctiveness metrics at signature resolutions of 5, 6, and 8 are given in Table 3.6.

Measure	Res.	R	P	U sig.	H sig.
DEO	5×5	1.7	0.94	< 0.01	< 0.01
(SigEnt)	6×6	1.6	0.93	< 0.01	< 0.01
	8×8	1.5	0.93	< 0.01	< 0.01
SGD	5×5	1.3	0.66	< 0.01	< 0.01
(NoGrad)	6×6	1.6	0.74	< 0.01	< 0.01
	8×8	1.9	0.87	< 0.01	< 0.01
ZC	5×5	8.9	0.94	< 0.01	< 0.01
(SigVar)	6×6	9.3	0.94	< 0.01	< 0.01
	8×8	7.5	0.94	< 0.01	< 0.01
ESTR	5×5	39	1.0	< 0.01	< 0.01
(SigVar)	6×6	19	0.99	< 0.01	< 0.01
	8×8	13	0.99	< 0.01	< 0.01
TEX	5×5	5.3	0.99	< 0.01	< 0.01
(SigVar)	6×6	5.2	0.99	< 0.01	< 0.01
	8×8	4.1	0.99	< 0.01	< 0.01
TEX	5×5	3.2	0.99	< 0.01	< 0.01
(MeanVal)	6×6	6.7	0.99	< 0.01	< 0.01
	8×8	3.0	0.99	< 0.01	< 0.01

Table 3.6: Summary of hypothesis test results on the image corpus. R is the ratio of distinctiveness means; $P = Pr(D_i > D_j \mid M_i < \vartheta \wedge M_j > \vartheta)$; U significance is the probability of erroneously rejecting the hypothesis of equal medians, and H significance of the hypothesis of the same population. See text for a more detailed description of the tests.

When the results are examined, we see that the metrics presented here satisfy all the tests, confirming the impressions gotten from the plots in Figure 3.8. The larger distinctiveness mean is at least 1.3 times the smaller, which indicates that a difference is noticeable. For all measurement functions but SGD, $P > 0.9$ which indicates that it is overwhelmingly probable that unambiguous views can be distinguished from ambiguous ones. For SGD, the resolution of the signatures seems to play a significant role; one explanation is that at lower resolution, the regions that are averaged over are larger, and hence more likely to contain significant contrast edges, obscuring the existence of large regions with little contrast. In any case, while $P > 0.7$ is not overwhelmingly large, it may still be reasonable if SGD is used with other measurement functions.

The significance of the median and population tests is quite striking. So striking, in fact, that the figures were cross-checked in two ways: a random partitioning of the data into 'pseudo-ambiguous' and 'pseudo-unambiguous' was used and the tests were performed, and the results of the tests on non-meaningful distinctiveness metrics (eg, SigVar for DEO) were examined. Random partitioning produced, as would be expected, no hypothesis rejection at any reasonable significance level. Non-meaningful distinctiveness metrics occasionally checked out on one of the two tests, but (a) not as significantly as those accepted, (b) almost never passed both tests, and (c) never passed all four tests. This leads us to conclude that the distinctiveness metrics presented here are good predictors of scene ambiguity and hence, local distinctiveness maxima should be unambiguous. This claim is tested more directly below.

Stability

One way to evaluate the inherent stability of a distinctiveness metric is to plot distinctiveness versus viewpoint. If large maxima exist, then it can be assumed that stability can be achieved. Also, such plots help us get a feel for what distinctiveness actually measures. Plots for two positions in our laboratory are shown in Figures 3.9 and 3.10. The first salient feature of these plots is that they have wide maxima at a coarse scale, implying large stability regions. This is mostly not the case for DEO, which is largely constant, but since it doesn't jump about randomly it works well in conjunction with other measurements. Another feature of these graphs is that the locations of their maxima correlate well across different measurement types. These two facts lead us to conclude that our distinctiveness metrics contain enough information for viewpoint stabilization.

If we look more closely at the plots and refer back to Figure 3.7, we can make correspondences between computed distinctiveness maxima and the physical contents of our lab. First, consider the doorway camera position. There are two large maxima here: at about 160° and at about 350°. The first corresponds to the camera looking towards the chair and door in the atrium, a region of considerable interest compared to the blank walls surrounding it. The second has the camera looking at the robot

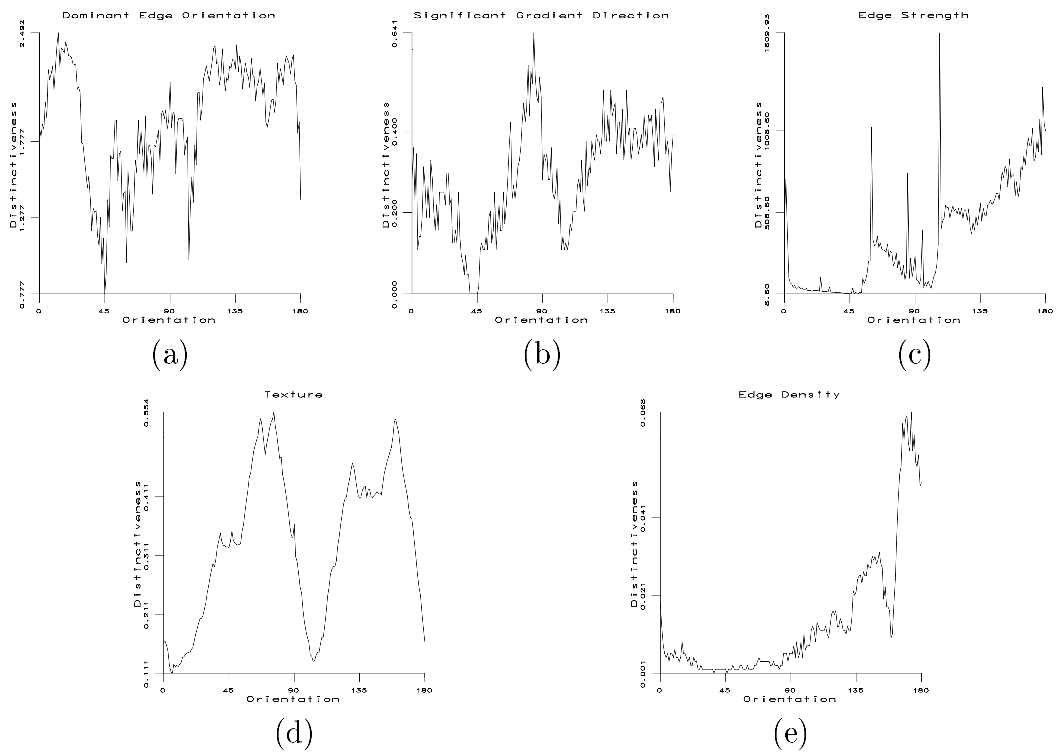


Figure 3.9: Distinctiveness (y-axis) plotted against camera orientation (x-axis) from the lab doorway. Orientation is given in units of 2 degrees (so '90' is really 180°). (a) Signature entropy of DEO. (b) Noticeable gradient on SGD. (c) Signature variance of ESTR. (d) Signature variance of TEX. (e) Signature variance of ZC.

arm, also a distinctive feature of the lab. The interior position gives, as we would expect, more distinctive viewpoints. We can pick out several: at 10°, at 100°, at 190°, and at 300°. The first corresponds to the Sun workstation[6] and workbench in the lower left. The second is the robot arm again. The third is a combination of the open door and the camera tripod. The last is the constellation of chairs and workstations in the lower right. Thus we see that signature distinctiveness corresponds quite well to our intuitive notions of 'distinctive views'.

[6]Sun is a registered trademark of Sun Microsystems Incorporated.

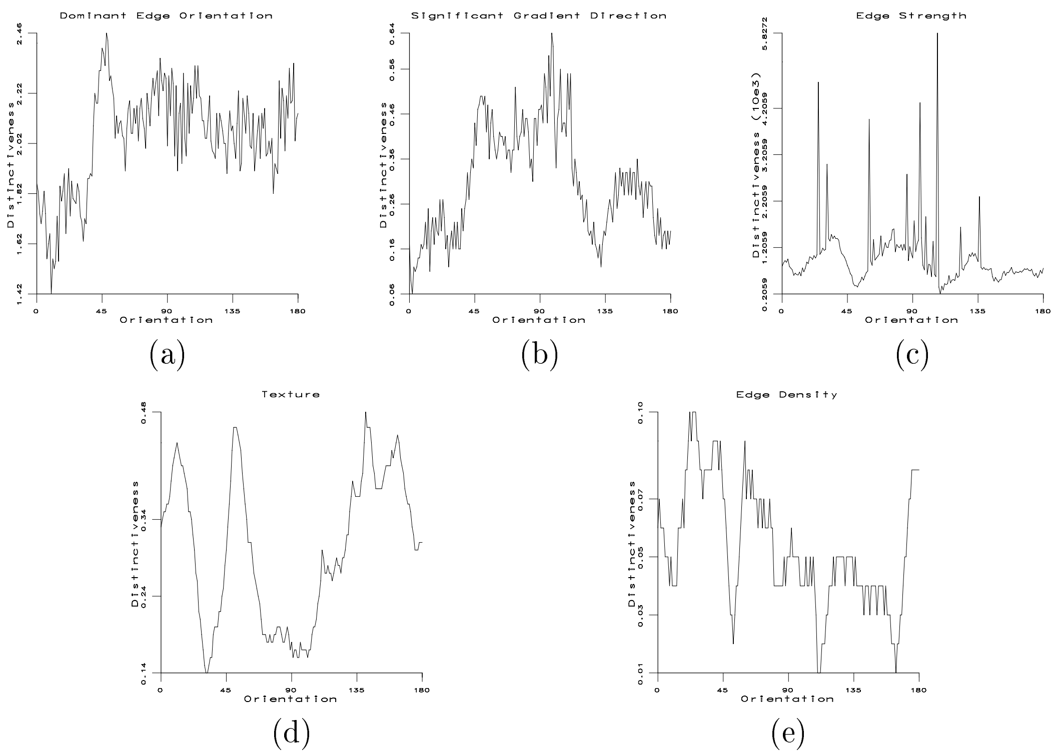


Figure 3.10: Distinctiveness (y-axis) plotted against camera orientation (x-axis) from the interior of our lab. Orientation is given in units of 2 degrees as above. (a) Signature entropy of DEO. (b) Noticeable gradient on SGD. (c) Signature variance of ESTR. (d) Signature variance of TEX. (e) Signature variance of ZC (jagginess due to plot quantization).

Dynamic experiments

We conducted experiments to measure several features of the distinctiveness maximizing algorithm described above in Section 3.3.3. The camera was mounted on an experimental pan-tilt platform manufactured by Zebra Robotics. The platform was positioned about 1 meter off the ground. For all experiments but one (as noted below), an 8mm lens was used. Two main positions were used, one in the middle of our lab (InLab) and the other in one of the lab doorways (Door). Most of the experiments consist of a rotational test suite, each test performed starting with a different camera orientation, at 15° intervals. The search parameters were $V_0 = 100°$ and $\epsilon_V = 5°$.

Figure 3.11: Zebra pan-tilt platform with CCD camera.

Distinctiveness and rotational stability

We first checked the distinctiveness values of the views found by trisection search. At each of the test positions the algorithm, using all five measurement functions, was used to find a local distinctiveness maximum. We then recorded the distinctiveness of the final viewpoint (an *attractive viewpoint*) with respect to all five measures. This was repeated for all starting orientations 15° apart.

A similar method was used to evaluate rotational stability for both the hillclimbing and trisection search methods. The same procedures were used, and the final orientations (*attractors*) after each search were recorded. If the method is rotationally stable, we expect the camera to home in on one of a small set of distinctiveness maxima, regardless of starting orientation. For our purposes, if two final orientations were within ϵ_V of each other, they were considered the same. We measure stability in two ways. First, we look at the total number of attractors found (A); second, we check the minimum number of attractors needed to account for half of the starting orientations (M). For a very stable method, both of these numbers should be small, relative to the number of tests run (36). But if M is small, then even if a few starting orientations lead to anomalous attractors, the method can still be considered usually stable.

Distinctiveness test results are given in Table 3.7. The first feature of the data which is noticed is the significant difference between distinctiveness of InLab and

InLab					Door				
DEO	SGD ($\times 10^2$)	ESTR	TEX ($\times 10^2$)	ZC ($\times 10^3$)	DEO	SGD ($\times 10^2$)	ESTR	TEX ($\times 10^2$)	ZC ($\times 10^3$)
2.1	31	911	32	24	1.7	14	1129	25	25
2.0	50	277	23	25	1.8	22	1305	32	17
2.1	55	434	22	8	1.8	25	1421	31	17
2.1	47	1060	27	32	1.9	2	58	10	4
2.0	25	1031	33	32	1.9	3	83	8	3
1.8	27	1455	36	21	1.9	11	56	16	3
2.2	31	861	30	27	1.9	13	90	20	4
2.1	34	999	38	13	2.1	17	101	17	5
1.7	27	1203	32	29	1.7	28	104	15	5
2.0	12	1259	27	40	1.9	8	72	23	5
1.9	23	1146	25	33	2.3	26	467	20	8
2.2	34	1422	26	13	2.2	30	622	25	7
2.2	36	1291	28	14	2.3	34	692	26	9
					2.3	30	986	37	6
					2.3	38	1108	41	6
Average					**Average**				
2.1	35	1045	29	24	2.0	20	552	22	11

Table 3.7: Results of the rotation test experiment for InLab and Door, using trisection. The calculated distinctiveness of attractor views are shown for the five measurement functions. Averages were computed over all starting orientations.

Door attractors. This can be explained by the fact that the doorway looks out on a hallway, thus many of the attractors (being *local* maxima) are nondescript views of the hall. If the values are compared against the plots in Figure 3.8, the distinctiveness of most attractors are clearly reasonable, and often excellent. This argues that the local maxima found by our method tend to be truly distinctive. Another thing to notice is that the distinctiveness measures are sometimes complementary, in that attractors distinctive in, say, SGD, are less distinctive in ESTR, and vice versa.

Table 3.8 summarizes the results of the rotational stability tests. The stability results show a reasonable level of stability, in that the majority of starting orientations go to one of 4 or 5 attractors. On the other hand, the total number of attractors is over a third of the number of tests performed. This can probably be attributed to two features of our testing methods. First, only attractive orientations within 10° were

Search		**InLab**	**Door**
Trisection	Attractors	13	15
	Main attractors	4	5
	Average steps	9	9
Hillclimbing	Attractors	14	16
	Main attractors	4	4
	Average steps	12	11

Table 3.8: Rotational stability and efficiency results. Main attractors are the minimal set attracting most viewpoints.

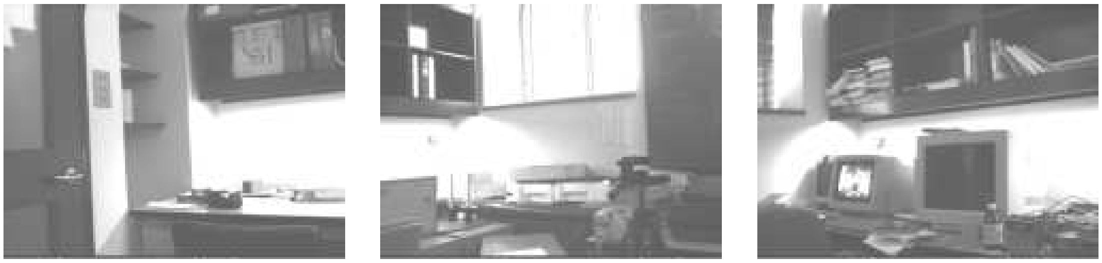
Search	Direction	0cm	10cm	20cm	30cm	40cm
Trisection	Forward	46°	47°	48°	60°	62°
	Transverse	47°	61°	48°	56°	45°
Hillclimbing	Forward	65°	50°	-161°	75°	—
	Transverse	95°	30°	30°	180°	—

Table 3.9: Orientations of the attractors in the translational stability test.

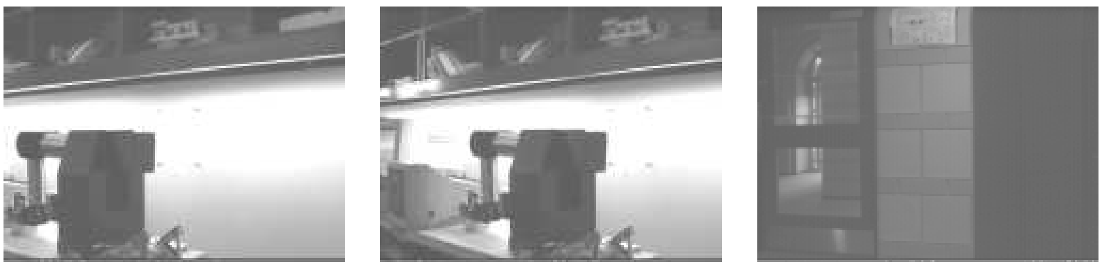
considered the same. However, with the camera's 36° field of view, viewpoints up to 18° apart will be matchable (using offset matching, see Section 3.2.2); the 'within ϵ_V' criterion, while justifiable, is perhaps too strict. Second, we would expect larger values of V_0 and smaller values of ϵ_V to produce more stable attractors, since (a) it is less likely for an attractor to be cut off from the initial interval, and (b) local maxima are found more precisely. There is clearly a tradeoff between search time required and quality of the solution. We also note that hillclimbing and trisection search have similar stability properties, though hillclimbing is more expensive.

Translational stability

The other sort of stability is with respect to viewpoint translation. We tested stability for forward and transverse translation (with respect to the starting viewing direction) separately. For each direction, we ran the trisection algorithm at 5 positions at



InLab



Door

Figure 3.12: The images at the three most frequent trisection-attractive viewpoints for InLab and Door in the rotation test.

10cm intervals. A translationally stable method should give rise to similar attractive viewpoints and matchable views at different positions, in the same way as rotational stability implies for different angles. The results are summarized in Table 3.9. The data show a fair measure of attractor stability for trisection over a translation range of 40cm. While two attractors seem to have emerged, looking at Table 3.9, the strictness of the 'same attractor' criterion can be seen to have an effect, as the images in Figure 3.13 show. The images in each sequence, while not identical, show nearly the same scene; nearly enough to match at a signature offset. Thus trisection appears to produce reasonable translational stability. However, hillclimbing does not do nearly as well. This is because the method uses a narrow window for deciding when a point is a maximum, and hence is more sensitive to low-amplitude distinctiveness oscillation (visible in Figures 3.9 and 3.10). This means that a small change in starting orientation can cause the method to miss a previously-found maximum. This reflects

0cm 10cm 20cm 30cm 40cm

(a)

0cm 10cm 20cm 30cm 40cm

(b)

Figure 3.13: Images of the attractive viewpoints for the translational stability test, using trisection. (a) Forward sequence. (b) Transverse sequence.

a more general problem, that of *distinctiveness smoothing*. Ideally, we would like to perform a low-pass filter on distinctiveness to produce a truly stable method; however, this is difficult to achieve on-line. This problem definitely requires further study.

Ambiguity

We also tested trisection attractors' ambiguity directly, using the image corpus. The camera was configured with a 25mm lens and set at a height of 110cm (to match the corpus). The camera on the pan-tilt head was placed at Door, which was also a position used in the generation of the image corpus. A rotational test suite at intervals of 20° was performed, and the attractive viewpoints were matched to the corpus, using all five measurement functions, to directly evaluate their ambiguity. The results are given in Table 3.10. The difference is quite striking; the average regular viewpoint is ambiguous, while the average attractor is not. In fact, only one attractor was judged to be ambiguous; at least one is to be expected in a doorway,

	Maximum		Average		Not amb.
Starting	110	38%	20	7%	14
Attractive	21	7%	2.7	1%	17

Table 3.10: Results of the ambiguity test. Shown are the numbers of images matched in the corpus by images at all starting orientations (at 20° intervals) and at the resultant attractive orientations. We show the maximum number of images matched in each set, the average over the set, and the number of views in the set (18 images) not judged ambiguous (by the $< 5\%$ criterion). We also give the percentage of the image corpus (291 images) matched.

as noted above. This reinforces the conclusion that these distinctiveness metrics are good predictors of ambiguity.

3.5 Related Work

Image-based recognition

Nelson's work on image-based homing [Nelson, 1989] used a coarse pattern-matching approach, which our framework generalizes. His 'patterns' are essentially image signatures based on measurements of dominant edge orientation; he used them to construct a reactive plan for homing a robot to a predetermined location. The image signature method described here generalizes this notion to encompass matching based on multiple feature types, at different offsets, and at multiple scales. In a similar vein, Horswill [1993] uses coarse grey-scale intensity matching for place recognition. Another similar approach is taken by Zetsche and Caelli [1989] for the problem of invariant pattern recognition. They use oriented gaussian filters to derive a 4D translation-, scale-, and rotation-invariant representation of 2D input patterns, using cross-correlation for matching.

A different image-based technique is described in [Hong *et al.*, 1991]. The authors use a spherical mirror to obtain a 360° image of the robot's surroundings. They then segment the image's horizon circle, finding a discrete set of 'landmarks'. Landmarks are described by a window of image intensities, locally normalized for illumination. If the orientation of the robot is assumed known (to within a small tolerance) these landmark 'signatures' can be matched to one another to recognize a location, or to perform local image-based homing. This homing technique is more general than Nelson's, since it does not require that the robot visit all points in its environment. This approach also bears some similarity to the landmark-based approach of Levitt *et al.* [1987], although they do not give a detailed account of landmark extraction and matching.

A third sort image-based recognition approach is taken by Swain [1990], who uses color cues for object recognition. The method is based on the use of color histograms, which are obtained by counting the number of times each color (red/green/blue value) occurs in the image. Matching is done using histogram intersection, which finds the fraction of model pixels for which an image pixel can be found with the same color. Localization of the recognized object can be done approximately, by histogram backprojection (see [Swain, 1990] for details). The method is simple, efficient, and reliable, and could easily be incorporated as a measurement function in an image signature library.

Geometric techniques

Another approach to place recognition for topological maps is to use local geometry. Sarachik [1989] has developed a robust method for estimating the size and aspect ratio of rectangular rooms. In a non-homogeneous environment, this could be useful for place recognition. Kriegman [1989] developed a system for instantiating generic place models (such as hallways), which can be used both for recognizing place categories, and for matching based on high-level place descriptions. Moravec and Elfes [1985] used correlation of sonar-derived certainty grids for rough place recognition and robot localization. Braunegg [1990] developed a method for matching place descriptions based on the relative 3D positions of vertical edges. Atiya and Hager [1991] developed a system for precise robot localization using stereo measurements of vertical edge segments. They solve correspondence by matching 3D geometric invariants. In a similar vein, Leonard and Durrant-Whyte developed a sonar-based mapping system [Leonard *et al.*, 1992], which effectively performs localization using sparsely distributed geometric features in a Kalman filtering framework.

3.6 Discussion

Image-based place recognition has several benefits over reconstructionist approaches. In particular, the method of image signatures is simple and easy to apply, generally applicable to all sorts of environments, computationally inexpensive, and is easily integrated with other recognition cues. When used with appropriate measurement functions, image signatures support image-based recognition over a range of viewer positions and orientations. The qualitative performance of different measurement functions can be evaluated by the twin metrics of stability and distinguishability, so that appropriate measures can be found for particular environments. Signatures derived from different measurements can be used in combination for matching, to improve correctness; ambiguity filtering can also be applied.

We have also identified a difficulty for any image-based recognition framework—the *recognition problem*, which consists of two coupled subproblems: the ambiguity problem and the stability problem. We deal here with this by problem using viewpoint motion. Scene ambiguity, at least in our indoor environment, appears to be reasonably well predictable from image structure by simple distinctiveness metrics. Using such metrics, a local search procedure can be used to efficiently find locally distinctive viewpoints. Most of the time, these locally distinctive viewpoints are also unambiguous, and hence useful for recognition. It also turns out that there are not many such attractive viewpoints, and so the method ensures a certain amount of viewpoint stability. This improves recognition in that the same viewpoint will usually be found each time a place is visited, easing the necessity of storing very large numbers of image signatures.

Chapter 4

Navigational Representation

Journey over all the universe in a map,
without the expense and fatigue of traveling,
without suffering the inconveniences of
heat, cold, hunger, and thirst.
—Cervantes

Summary

This chapter describes diktiometric representation, which combines topological, geometric, and behavioral information into a single representation, meant to be useful for robotic navigation. To that end, the basis of diktiometry is the waypoint, which serves both as a nexus for intersecting paths (so that they can be combined) and as a localization point (so the robot can know where it is). The coarse path structure of the environment is represented by the topological relationships between waypoints.

Finer path structure is expressed by the robot's behavior in going between waypoints, and so is represented by control programs expressing that behavior. Since these plans use sensory feedback, they implicitly represent the environmental structure between waypoints. These plans are derived from observations of the robot's behavior, encoded as effector module events. The plans that are thus derived make use of the stability of the effector modules, and so are robust to a variety of environmental changes.

In addition to this qualitative path information, we also need to represent geometric information about the shape of the path graph. We identify two geometric representation issues: referential locality and uncertainty representation. Referential locality can be treated in a uniform framework by considering reference graphs, which encode all locally known relations between points. There is a tradeoff between efficiency and precision; more local representations are more precise, but less efficient. Geometric uncertainty can be represented using either statistical or set-based methods. Set-based methods avoid making strong distributional assumptions, which are rarely satisfied exactly in robotics. The use of intervals as a set representation technique provides a simple and efficient way of dealing with geometric uncertainty, as we saw above. The difficulty with using intervals is that it can lead to overestimation when rotation is introduced. This overestimation can be compensated for somewhat by using more complex representations; their utility remains to be seen (we found regular intervals adequate). Rotational uncertainty also causes problems for statistical techniques due to nonlinearities, so achieving accurate orientation information appears to be key for any robotic map-building using geometric information.

4.1 Representation and Behavior

The function of a mapping system is to observe the world around an agent and build mental maps of that world. The world is a complex place, and many different sorts of representations of it may be constructed, depending on what aspects of the world are considered important and attended to (the 'subjective world'). Even once this choice is made, a code for representing the structure of the subjective world must be developed—there are many ways to skin a pomegranate. As we discussed earlier, mapping does not occur in a vacuum, and must be considered in the context of how it will be used. The function of mapping is to support navigational planning. Hence, the representation scheme we use must be designed to support navigation.

In our architecture, navigation can be viewed as giving directions to an intelligent, but ignorant deliberator. Given that, what sort of representations are useful for navigation? Let us examine the directions to my home that I might give someone unfamiliar with New Haven:

> To get to my home from Watson Hall, go down Prospect Street and make a right onto Grove Street. Follow it until you get to a chaotic intersection. There, you want to get on Whalley Avenue—stay in the middle lane and take the far right turn (not the closer right turn which is Dixwell). Stay on Whalley for a bit over a mile until you get to a large intersection which is Boulevard; there's a Dunkin Donuts on the corner. If you hit the cemetery, you've gone too far. Make a left onto Boulevard and take the second right (the one *after* the light), which will be Maple Street. It's the second blue house on the right.

There are a number of interesting features of these directions. First is the fact that none of the instructions are very detailed. They assume low-level procedural knowledge ("turn right"), as well as relying on environmental structure ("Follow it") and perceptual feedback ("there's a Dunkin Donuts on the corner"). If we look more closely at the details of the directions, more structure emerges. The directions can be broken into segments, each of which ends at a specific location ("large intersection"). These locations are described well enough for them to be recognized, but not necessarily visualized ("chaotic intersection"). Also, directions are generally specified locally, with respect to landmarks or environmental structure ("the far right turn"). We also see the use of landmarks for disambiguating percepts ("not the closer right...") and for diagnosing run-time plan failure ("If you hit the cemetery, you've gone too far").

This chapter describes diktiometric representation, which integrates topological, geometric, and behavioral information into a single representation. The basis of diktiometry is the waypoint, which serves both as a nexus for intersecting paths (so that they can be combined) and as a localization point (so the robot can know where it is). The coarse path structure of the environment is represented by the topological relationships between waypoints. Finer path structure is expressed by the robot's behavior is going between waypoints, and so is represented by plans expressing that behavior. Since these plans use sensory feedback, they implicitly represent the environmental structure between waypoints.

In addition to this qualitative path information, we also need to represent geometric information about the shape of the path graph. We identify two geometric representation issues: referential locality and uncertainty representation. Referential locality can be treated in a uniform framework by considering reference graphs, which encode all locally known relations between points. There is a tradeoff between efficiency and precision; more local representations are more precise, but less efficient. Geometric uncertainty can be represented using either statistical or set-based methods. Set-based methods avoid making strong distributional assumptions, which are rarely satisfied exactly in robotics. The use of intervals as a set representation technique provides a simple and efficient way of dealing with geometric uncertainty. The difficulty with using intervals is that it can lead to overestimation when rotation is introduced. This overestimation can be compensated for somewhat by using more complex representations; their utility remains to be seen (we found regular intervals adequate for our task). Rotational uncertainty also causes problems for statistical techniques due to nonlinearities (as noted by [Smith *et al.*, 1986]), so achieving accurate orientation information appears to be key for any robotic map-building using geometric information.

In the following section, we will develop a basis for navigational representation based on the notion of the *waypoint*. In Section 4.3, we will elaborate the data structures required to capture the overall structure of the environment, using topological and metric information. We then discuss in the next section a detailed approach to representing metric information. The last part of the puzzle is developed in Section 4.5, where we develop a method for directly representing robot behavior.

4.2 A Basis For Representation

Lynch [1960] distinguishes five types of primitive elements which combine to form a representation or 'image' of a city or other structured space: paths, nodes, landmarks, edges, and districts. Paths represent corridors along which people can move, effectively constraining motion to one dimension. Nodes are specific points of interest, such as plazas or important buildings. They are generally places where multiple paths come together, and so are important for navigation. Landmarks are those distinctive artifacts or topographic configurations that are visible from some distance and can be used for localization and navigation. Edges represent barriers (sometimes partial) to travel, and are also often boundaries between different regions. Districts are extended regions in a city, characterized in some recognizable way, either structurally, perceptually, or functionally.

4.2.1 Paths and waypoints

Our work follows Kuipers and Byun [1988], and only considers the first two sorts of representational elements, paths and nodes. This is partially motivated by our choice of indoor environments as a domain, where paths and nodes tend to dominate. Landmarks, in particular, are less useful indoors than out, since the view of the observer is more constrained. For example, the system developed by Levitt *et al.* [1987] is only appropriate for outdoor use. Furthermore, landmark use involves the difficult problems of segmentation and object recognition, not to mention the difficulty of deciding what should be considered a landmark. Using edges similarly requires complex perceptual abilities, in order to detect and recognize extended structures at a small number of viewpoints. Finally, districts, as used by people, often have vague or nonexistent boundaries (as noted by Lynch), and may only be distinguishable by complicated semantic criteria.

The robot's navigational knowledge may be summarized as the set of all the paths it has ever taken between places. Consider then the problem of synthesizing navigational plans for new goals that the robot has not attempted before. Synthesizing such plans from knowledge of previously traversed paths can be done by using *waypoints*. A waypoint is a location in the world, such as a corridor intersection or a courtyard, which can be reliably distinguished from its surroundings, both locally (I can tell when I'm at a waypoint) and from a distance (I can go to a nearby waypoint). Paths which pass through waypoints can be broken up into waypoint-to-waypoint segments, which can then be strung together in many different ways, allowing great flexibility in plan synthesis. In addition, different path segments may be more or less reliable, or only traversable at certain times; being able to transparently switch between different paths from A to B at intermediate points can be a great advantage. Waypoints constitute *nodes* in Lynch's [1960] sense. Waypoints are particularly ubiquitous in human-designed environments, almost certainly in order to improve their navigability[1].

In Chapter 2 we discussed the model of waypoints assumed in this work. The relevant properties of waypoints for the purposes of this chapter are: waypoints are typed; they

[1]Perfect navigability is not always achieved, due to the interference of other factors. If waypoints are not easily distinguished, or if there is no recognizable large-scale structure to a space, it can take a very long time to learn to navigate in a space. Some of these issues are discussed in Norman's work on cognitive design [Norman, 1980].

can be recognized and approached from afar; and they are orientable. Typing means that we can distinguish waypoints based on different sorts of locally stereotyped environmental structure (we can tell a raven from a writing-desk). Recognition is crucial—the robot must be able to reliably tell if it is at a waypoint or not, otherwise waypoints would be useless for localization. Waypoint approach is also important for localization, since it serves to 'actively' localize the robot. Finally, waypoints define local coordinate frames, relative to which the robot sense its orientation. This ability, combined with a knowledge of the waypoint's own orientation (learned, of course; see Chapter 5), allows the robot to orient itself more globally.

4.2.2 Waypoints, distinctive places, and gateways

Our conception of waypoints is similar to other notions of discrete point-like places used in the literature, primarily the *distinctive places* (DPs) of Kuipers and Byun [1988] and the *gateways* of Kortenkamp *et al.* [1992]. A DP is defined as a robot location which maximizes some *distinctiveness measure*, a function on robot sensor readings. One distinctiveness measure used by Byun [1990] is maximized when nearby obstacles are equidistant. This measure can be used to advantage in the vicinity of corridor intersections, for example. While DPs may be primarily thought of as locally recognizable robot positions, Kortenkamp's gateways are conceived as positions which separate qualitatively different spaces in the environment. The canonical example of a gateway is a doorway. Gateways are detected by sensing changes in the sensory quality of the surrounding space—by noticing a change in the robot's view, or by detecting openings or constrictions around the robot. Since gateways by their very nature restrict the local motion of the robot, gateways have canonical directions, which are used to constrain perception for recognition.

The differences between waypoints, DPs, and gateways are subtle, but important. Since DPs are defined entirely in terms of local optimization of distinctiveness measures, long-range detection and approach is impossible in general. The use of designation and approach obviates this problem for waypoints. Long-range detection of waypoints also means that the robot need not be at a waypoint in order to locate itself[2]. DPs also do not have any notion of local orientation. The uniform mathe-

[2]This would mean treating waypoints as landmarks. While this is not implemented in our current mapping system, it poses no conceptual difficulties for our framework.

matical basis of DPs might permit a more precise analysis of overall system behavior, but that still remains to be seen.

Gateways are very similar to waypoints, and in fact, gateways may be considered a type of waypoint. The chief difference in formulation is that by basing the notion of gateway entirely on its role as a transition point, only local gateway detection is possible. For this reason, even local approach (as for DPs) cannot be done, since a change in view can only be determined by comparing views from both sides of a proposed gateway. In fact, it may only be possible to tell when a gateway has been encountered *after* the robot has passed though it. Gateways do, however, form an important class of waypoints, in that they serve an important semantic function in our organization of space.

4.3 Diktiometry

Representing waypoints and the paths between them is done most simply as a directed graph, the *path graph*, whose nodes represent waypoints and whose arcs represent direct paths between waypoints. Such graphs represent the connectivity of the environment well, so they have been used in many different sorts of map-learning research. In particular, map-learning may be formalized in this framework as inferring minimal finite-state automata from observations [Rivest and Schapire, 1989]. Perceptual noise and effector error can be modeled by making state observations and transitions probabilistic (as in [Basye *et al.*, 1989] and [Dean *et al.*, 1992]). Navigational planning in a purely graph-based framework can be done easily by searching for a path of arcs through the path graph between the starting node and goal, possibly optimizing some measure of performance (shortest path). This simplicity depends upon some assumptions about the capabilities of low-level perception and action. In our execution framework, these assumptions are those about the properties of effector modules, specifically stability and cognizant failure (see Chapter 2). The stability of effector modules means that we never need to localize the robot *precisely* with respect to its environment. As long as we are 'close enough', the effector modules will do the rest. In particular, waypoint approachers allow the robot to localize itself precisely by moving the robot repeatably to the same position (this also forms the basis for [Kuipers and Byun, 1988]). However, since absolute stability is rarely achievable in the real world, we require that modules fail cognizantly whenever possible, so that we know that they failed and can replan accordingly.

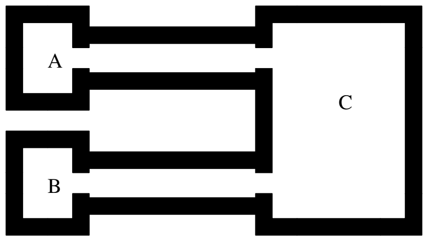


Figure 4.1: An example of local ambiguity.

4.3.1 Topology and geometry

The path graph represents certain topological properties of the environment, namely the direct connections between waypoints. However, these topological properties provide only weak constraints for recognizing individual waypoints, particularly during map construction, when few relationships are known. In the world depicted in Figure 4.1, for example, rooms A and B cannot be distinguished by local sensor readings. Furthermore, if the robot cannot always tell which doorway it went through when leaving room C (quite likely, given present technology), the two cannot be reliably distinguished by topological inference either. This means that recognition must be done purely on the basis of local perceptual cues, which are often ambiguous and unreliable. However, another form of non-local information for recognition is geometry. Knowing something about the positions of waypoints with respect to one another provides much stronger constraints for recognition (and sometimes planning) than topological relations alone. We term a representation which combines a path graph with relative geometric information, a *diktiometry*[3].

[3]From the Greek δίκυον, meaning 'network', and μέτρον, meaning 'measure'. A diktiometry is thus a 'measured network'.

4.3.2 Data structures

To be more specific, a diktiometry consists of two graphs: a *path graph*, and a *reference graph*. The path graph's nodes represent waypoints, and its arcs represent actions moving between them. Each waypoint node is labeled with its waypoint type, a set of views describing percepts visible from the waypoint (for recognition), and some geometric information. This includes a position and orientation estimate as well as some ancillary data used for error-correction, described in Chapter 5. Action arcs are labeled with the actions which traverse them and an estimate of the relative position and orientation of the robot at the ends of the arc. Action representation is discussed in Section 4.5 below. The structure and use of the reference graph is described in the next section.

4.4 Metric Representation

As we discussed, topological relations between waypoints are not sufficient to support passive map-learning, so we use geometric knowledge as well. This introduces the important problem of representing geometric uncertainty, since geometric relations will never be known with complete accuracy. This section addresses this *metric representation problem* by formalizing it as a problem of representing uncertain geometric relationships between points. The abstract points used in this section may be identified with waypoint nodes in the path graph. We develop representations and algorithms for computing relations between points and incorporating new measurements. For the reasons discussed in Section 2.4.3, we choose to use a set-based uncertainty representation rather than a statistical method. Note however that such methods (like that of Smith *et al.* [1986]) can be incorporated into our representational framework, if they are appropriate in a particular domain.

4.4.1 The metric representation problem

We formalize our geometric representation problem abstractly by considering a set of points in the world, $\{p_i\}$. These points represent the positions and orientations of waypoints, though they could just as well describe visual landmarks or objects.

The points exist in some known configuration space. For purposes of mobile robot mapping we take the space of 2D positions plus rotations as our configuration space; our methods are easily generalized. At each time step k, we are given a measurement of the relation between two points, $z_{ij}^k = p_j - p_i + v^k$, corrupted by noise v^k. The model for the observation noise v^k depends on the representation method used (here set-based), but abstractly, we can associate an *uncertainty parameter* U_{ij}^k with each observation. Such observations may be thought of as constraints on the relative positions of points. The task of *metric representation* is to represent the constraints provided by these observations, under a given uncertainty model, in a data structure providing the operations:

- `add-constraint`$(\langle z_{ij}^k, U_{ij}^k \rangle)$ adds the information from observation k into the data structure;
- `relate`$(p_i, p_j) = \langle \hat{x}_{ij}, \hat{U}_{ij} \rangle$ returns the best estimate of the metric relationship between p_i and p_j with the estimated uncertainty of that estimate. We also term this operation *constraint retrieval*.

It is assumed that we know which points each observation relates; this *correspondence problem* is addressable only in the context of the full mapping system. To aid in determining correspondence, though, we also require that an uncertainty model provide the operation:

- `consistent?`$(\langle x_{ij}^0, U_{ij}^0 \rangle, \langle x_{ij}^1, U_{ij}^1 \rangle)$ returns true if two hypothesized relations between a pair of points are consistent with each other.

4.4.2 Issues in metric representation

There are a few specific issues that must be addressed in order to incorporate geometric information into our map representation; the questions largely concern the topic of *uncertainty management*. Quantitative uncertainty arises in robotic applications because sensors and effectors are imperfect. Sensor readings do not accurately reflect reality, and effectors never do exactly what they should. In particular, we are interested here in measuring the robot's changes in position as it moves through the world, so that we know where waypoints are. Such measurements are based on odometry and visual feedback, which are prone to a variety of errors (see Section 2.4.3 for a detailed discussion).

There are two different sorts of issues we need to discuss here. The first is the question of using global versus local frames of reference. In the global approach, all positions are represented with respect to a single point in the world, the *global reference frame*. The alternative is to use multiple such reference frames for positions in different areas of the world. In the limit, each position stored may be considered a frame, and all relations between all points are kept. This is the approach discussed in this chapter; a more general framework for varying representational locality is left for future work. The question of *referential locality* has important theoretical import for uncertainty management, as well as for computational complexity. Proper use of local reference frames can significantly improve efficiency (with some information loss).

The other main issue is how metric uncertainty is to be represented. Statistical techniques represent parameter uncertainty as probability distributions over possible values, using Bayes' theorem to combine observations. Set-based methods, on the other hand, use bounds on parameter values to represent all possible values consistent with observed values. The effects of using these techniques differ by virtue of the fundamentally different assumptions that they make about the nature of uncertainty. This dissertation uses a set-based approach to metric representation, as a system design decision. However, the place-recognition and map-making methods developed here should be adaptable to statistical representation methods (such as that of [Smith *et al.*, 1986]) as well, should they be preferable in a given domain. Our set-based approach, and the algorithms that support it, are described below in Section 4.4.4.

4.4.3 Reference graphs

We can represent a set of metric constraints (as above) by a graph, whose nodes represent the points p_i and whose links are labeled with the stored relations $\langle x_{ij}, U_{ij} \rangle$:

Definition 4.1 *A* reference graph *is a simple directed graph G, whose vertices represent points p_i, and each of whose links $e = \langle p_i, p_j \rangle$ is labeled with a metric relation rel$(e) = \langle x_{ij}, U_{ij} \rangle$. We require that links be symmetric: $x_{ij} = -x_{ji}$ and $U_{ij} = U_{ji}$.*

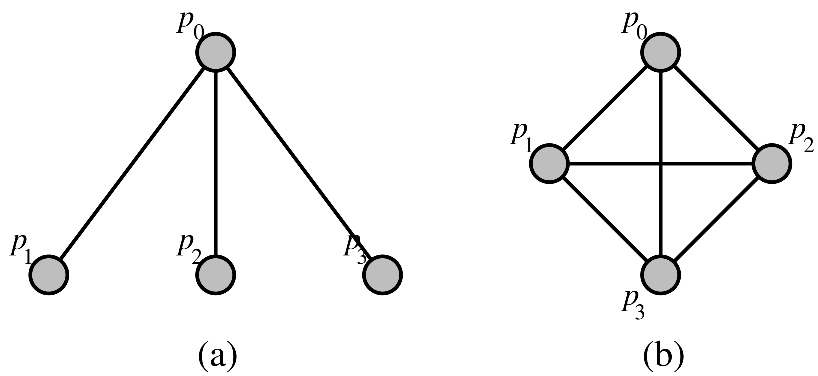


Figure 4.2: Reference graphs for four points. (a) Global reference frame case. (b) Completely local case.

The set of vertices (points) of G are referred to as points(G). We denote the number of points in G by $|G|$. A complete reference graph *is a reference graph with links between all pairs of points.*

Now consider the two extreme types of representing metric constraints discussed above: the global reference frame approach and the completely local approach. The extreme reference graphs for four points are shown in Figure 4.2. The locality of the representation is given in the structure of the graph; this structure determines a position on the 'complexity/information loss' tradeoff axis. Algorithms for incorporating new observations and extracting relations can all be defined as algorithms on reference graphs, conditioned on the type of uncertainty representation used. We develop such algorithms further below for set-based representations.

Referential locality

The problem of *referential locality* is an important one when representing geometrical information with uncertainty (see, e.g, [McDermott and Davis, 1984]). Consider two possible methods for representing the geometric information contained in the observations $\{z_{ij}^k\}$. The first is to choose some distinguished point as p_0, and then represent estimates of positions of all other points with respect to that *global reference frame*. Formulating a representation strategy with global reference is fairly simple.

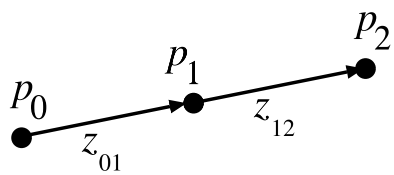


Figure 4.3: Simple example showing the nonlocality effect. $z_{01} = 5 \pm 2$, $z_{12} = 5 \pm 3$.

Also, incorporating new observations and extracting relation estimates both take constant time. However, information about uncertainty is lost in going to a global reference frame.

To illustrate the problem, we can use the example depicted in Figure 4.3. For this example, we will use a simple bounds representation of uncertainty. Suppose we decided to represent all points from the standpoint of p_0. After we get the observation z_{01}, we have `relate`(0,1) $= 5 \pm 2$. We then get $z_{12} = 5 \pm 3$ and incorporate it, getting, correctly, `relate`(0,2) $= 10 \pm 5$. However, since we only represent positions with respect to p_0, the specific information about x_{12} has been discarded. So if we now estimate the relationship of p_1 and p_2 from the global representation that we have constructed, we get

$$\begin{aligned} \texttt{relate}(1,2) &= \texttt{relate}(0,2) - \texttt{relate}(0,1) \\ &= (10 \pm 5) - (5 \pm 2) \\ &= 5 \pm 7 \end{aligned}$$

with uncertainty much higher than what we had from the single observation z_{12}. In going to a global frame of reference, we lose information regarding direct relations between points other than the origin. This may be termed the *nonlocality effect*. This effect causes subsequent computations of said relations to have higher uncertainty estimates, since all the uncertainty to the origin and back must be taken into account. In this paradigm, I might compute the distance from my workstation to my office door via Hong Kong, and get an answer of 3m±500 Km—a useless estimate, indeed! The only way to avoid this problem is to use a locally referential representation.

An approach diametrically opposed to the global one would be to store the best estimates of each relation from direct observations. That is, we maintain a set of variable pairs $\{\langle x_{ij}, U_{ij} \rangle\}$, where x_{ij} and U_{ij} are calculated solely from observations directly relating p_i and p_j, z_{ij}^k (for some set of k's). The best estimate of a particular relation $\langle \hat{x}_{ij}, \hat{U}_{ij} \rangle$ can then be calculated based on all the information at hand, so that no information is lost by going to a global reference frame. Local approaches of this sort, however, are much more expensive, of complexity at least cubic in the number of points. This complexity problem can be mitigated by a more global representation. Such techniques, however, necessarily throw away information; there is a tradeoff between information and efficiency which must be evaluated relative to the domain of application. In some cases, accuracy is paramount, and efficiency can be sacrificed; other domains have opposite requirements.

4.4.4 Set-based uncertainty

Before we can develop algorithms implementing `add-constraint` and `relate`, we must decide how to model geometric uncertainty. In Section 2.4.3 we argued for using a set-based representation for odometric uncertainty. Since odometry is our primary source of information about metric relations between waypoints, we will use set-based representation here.

In set-based representation, we consider sensor noise to be bounded, and hence to fall within some set of possible configurations U. This is a reasonable assumption in many cases (for example, see [Atiya and Hager, 1991]); even unbounded errors can be treated as bounded, if tails are sufficiently thin and robust techniques are used. In particular, our concern is with odometric error which can be modeled using set-based techniques, as discussed above in Section 2.4.3. The uncertainty in a parameter estimate is the largest set consistent with all observations; observations are combined by intersecting set estimates with observation sets. (See Appendix A for a more complete discussion of interval estimation.)

In our case, we need to represent estimates of relative positions and orientations. For each pair of points p_i and p_j, we can talk about a set of possible relative configurations, S_{ij}. Note that[4] $S_{ji} = -S_{ij}$. To simplify notation, we represent the kth observation of the relative position of points p_i and p_j by the set, $Z_{ij}^k = z_{ij}^k + U_{ij}^k$. Concatenating

[4]We define arithmetic operations for sets in the usual way by:

$$S_1 \mathrm{op} S_2 = \{x | x = x_1 \mathrm{op} x_2, x_1 \in S_1, x_2 \in S_2\}$$

.

relations between S_{ij} and S_{jk} is the set of concatenations of all possible relation pairs, $S_{ij} \oplus S_{jk}$. Combining two constraints on p_i and p_j is done by taking their intersection: $\hat{S}_{ij} = S_{ij} \cap S'_{ij}$. Two set estimates are consistent with one another when their intersection is non-empty. Assume for now that these operations (of addition and intersection) are constant time without considering how sets are represented. The effective representation of sets is discussed below in Section 4.4.5.

Consistency

The basic routine underlying both `relate` and `add-constraint` for sets is `consistency`, which computes the maximal consistent relation between each pair of points (using all observed relation estimates). To do this, we look at composite relations between points implied by known constraints along various paths between points.

Definition 4.2 *A* path relation S_π *along a path* $\pi = \langle p_{i_0}, p_{i_1}, \ldots, p_{i_m} \rangle$, *is defined as the concatenation of the pair relations along the path:*

$$S_\pi = S_{i_0 i_1} \oplus S_{i_1 i_2} \oplus \cdots \oplus S_{i_{m-1} i_m}$$

A path relation is just the relation between the path's endpoints implied by the constraints along the path.

Definition 4.3 *Given relation sets* S_{ij} *between pairs of points in* P*, a set* S *is* consistent *with a path* π *in* P *iff* $S \subset S_\pi$*. The* maximal consistent relation *between points* p_a *and* p_b *in* P *is the largest set* $\hat{S}_{ab}$ *consistent with all paths from* p_a *to* p_b *in* P*.*

The maximal consistent relation between two points is our best estimate of the points' relative configurations. Defining estimate correctness as path consistency enables us to avoid the intractability of the optimal relationship estimation problem noted by Davis [1984]. The result, proved by Yemini [1979], states that determining whether or not a set of points with distance constraints can be embedded in the plane is NP-complete. However, since this is due to the possibility of reflections of sets of points, which are disallowed in our framework, the result does not apply here. Constraints of the sort that cause trouble for Yemini and Davis would only result in very high uncertainty in our framework.

Another point to note is that we cannot use the standard $O(n^3)$ Floyd-Warshall all-points shortest path algorithm [Cormen *et al.*, 1992], since distributivity does not hold when we consider intervals of 2-dimensional positions plus rotations (see Appendix A).

It is easy to show that for any two points $p_a, p_b \in P$, $\hat{S}_{ab} = \bigcap_\pi S_\pi$, where π ranges over all acyclic paths from p_a to p_b (since loops don't add any new information). The fact that we only need to consider acyclic paths gives us the following path consistency algorithm.

Consistency($G = \langle P, \{S_{ij}\}\rangle$)

1. For all $i, j \in P$: $\hat{S}_{ij} \leftarrow S_{ij}$;
2. Repeat $\lceil \log_2 |P| \rceil$ times:
   (a) For all $i, j, k \in P$:
   $\hat{S}_{ij} \leftarrow \hat{S}_{ij} \cap (\hat{S}_{ik} \oplus \hat{S}_{kj})$;
3. Return $\{\hat{S}_{ij}\}$.

This algorithm clearly has complexity $O(n^3 \log n)$, where $n = |P|$, since all triples of points are considered on each of the $\log n$ iterations. To prove that the algorithm actually computes all the maximal consistent relations in G, we show that each relation $\hat{S}_{ij}$ that it computes is consistent with all acyclic paths from p_i to p_j.

Lemma 4.1 *If, for all $p_i, p_j \in P$, $\hat{S}_{ij}$ is consistent with all paths from p_i to p_j of length $\leq m$, then, for all $p_i, p_j \in P$:*

$$Q_{ij} = \bigcap_{p_k \in G} \hat{S}_{ik} \oplus \hat{S}_{kj}$$

is consistent with all paths from p_i to p_j of length $\leq 2m$.

Proof:

Consider an arbitrary point $p_k \in G$. Q_{ij} is clearly consistent with all paths going through p_k with $\leq m$ points before and after p_k. Since such a point can be chosen for an arbitrary path of length $\leq 2m$, Q_{ij} must be consistent with all paths of length $\leq 2m$. □

We can now prove the correctness of the `consistency` algorithm.

Theorem 4.1 (Consistency) *Upon completion of* `consistency`(G)*,* $\hat{S}_{ij}$ *is the maximal consistent relation between points* p_i *and* p_j *in* G*.*

Proof:

Let $\hat{S}_{ij}^t$ be the value of $\hat{S}_{ij}$ after the t-th iteration of step 2(a) (where $\hat{S}_{ij}^0 = S_{ij}$). We first show, by induction, that $\hat{S}_{ij}^t$ is consistent with all paths of length 2^t from p_i to p_j in G. Since $\hat{S}_{ij}^0 = S_{ij}$, $\hat{S}_{ij}^0$ is consistent with the path of length 1 from p_i to p_j. Suppose that $\hat{S}_{ij}^t$ is consistent with all length 2^t paths, for all p_i and p_j. Then, by associativity of intersection, we have that

$$\hat{S}_{ij}^{t+1} = \bigcap_{p_k \in G} \hat{S}_{ik}^t \oplus \hat{S}_{kj}^t$$

Now, from Lemma 4.1 and our inductive assumption, we have that $\hat{S}_{ij}^{t+1}$ is consistent with all length $2(2^t) = 2^{t+1}$ paths from p_i to p_j. Hence we conclude that $\hat{S}_{ij}^t$ is consistent with all paths of length 2^t from p_i to p_j. We can thus conclude that $\hat{S}_{ij}^{\log n}$ is consistent with all such paths of length n ($n = |G|$), hence all acyclic paths in G from p_i to p_j, and hence all paths between p_i and p_j. We can also see that $\hat{S}_{ij}^{t+1}$ is maximal, since were it any larger, it would be inconsistent with some $\hat{S}_{ik}^t \oplus \hat{S}_{kj}^t$, and hence some path from p_i to p_j. □

Retrieving and adding constraints

Given the algorithm described above for computing maximal consistent relations, we can now define the operations `complete-relate` and `add-constraint` in the set-based paradigm. If a complete reference graph is consistent, i.e, has an up-to-date set of $\hat{S}_{ij}$ labels, then `complete-relate`$(p_i, p_j) = \hat{S}_{ij}$, and is 'computed' by merely retrieving $\hat{S}_{ij}$. To ensure consistency, each call to `complete-relate` is preceded by a call to `consistency` on the relevant complete graph, if the last operation on that graph was an constraint addition. Thus, in the worst case, `complete-relate` can take $\mathrm{O}(n^3 \log n)$ time ($n = |G|$). Whenever the constraint between p_i and p_j is changed, a note must be made to run `consistency` before the next retrieval. If, however, the constraint is updated monotonically to a smaller uncertainty set (which is normally the case when an observation Z_{ij} is incorporated: $S_{ij} \leftarrow S_{ij} \cap Z_{ij}$), an $\mathrm{O}(n^2)$ incremental update suffices to keep the reference graph consistent. When S_{ij} is made smaller, we update the relation S_{kl} for every pair of points p_k and p_l by:

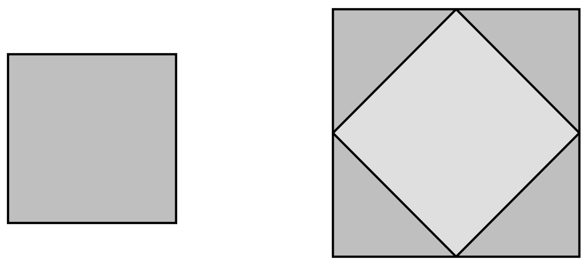
$$S_{kl} \leftarrow S_{kl} \cap (\hat{S}_{ki} \oplus S_{ij} \oplus \hat{S}_{jl}) \cap (\hat{S}_{kj} \oplus S_{ji} \oplus \hat{S}_{il})$$

4.4.5 Representing sets

In the discussion above, we assumed that sets were provided as a primitive object type. However, some limitation must be made on the class of sets considered, due to computational considerations. In this section, we will discuss and compare several different ways to represent sets. In particular, one important criterion is that set addition and intersection be constant time operations, in order to maintain the complexity results above.

Probably the simplest way to represent point sets is used by Brooks [1985]. Uncertainty is represented by a single tolerance parameter, and uncertainty sets by circles centered at a nominal position. Addition and intersection can easily be computed in constant time. However, there are two difficulties with this method. First, the intersection of two circles is not a circle; hence a bounding circle must be used, introducing severe overestimation. Second, circular sets cannot represent directional uncertainty; for example, we might expect odometric uncertainty of a robot moving down a hallway to increase much faster in the forward direction than in the transverse—this information cannot be represented by a single uncertainty radius. One general solution to these problems is to use convex polygons. The intersection of two convex polygons is always a convex polygon, and arbitrary sorts of directionality can be represented. The main difficulty with polygons is not representational power, but rather computational complexity. Since the intersection of two n-sided polygons can have as many as $2n$ sides, the space (and time) required grows exponentially in the worst case. This problem is only exacerbated for higher-dimensional polytopes.

One intermediate approach that has been applied in set-based estimation is use of bounding ellipses [Schweppe, 1968]. Directionality is preserved as for polygons, but since ellipses require only constant space, all operation are also constant time. However, as in the case of circles, intersections of ellipses are not ellipses, hence overestimation cannot be avoided when combining two constraints. On the other hand, in many cases (particularly if the ellipses are axis-aligned) the overestimation may be reasonably small.



(a) (b)

Figure 4.4: The effect of overestimation on axis-aligned intervals. (a) A 2-dimensional interval. (b) The best interval estimate (in dark) after rotating (a) by 45 degrees.

An approach with many similarities to the ellipse approach is the use of bounding intervals (Neumaier [1990] provides an excellent introduction). Uncertainty in each dimension is represented by an interval of values along an axis, producing a hyper-rectangle. Intersection is a constant-time operation for a given dimensionality, and has no inherent overestimation. The difficulty is with rotations. If the configuration space includes rotations, overestimation can become a severe problem when adding intervals (see Figure 4.4). This can be ameliorated by representing non-axis-aligned intervals (that is, arbitrary rectangles). This, however, introduces overestimation into intersection; this tradeoff must be evaluated for particular applications. In some cases, exact or nearly-exact set representations can be used during preliminary processing before database incorporation, to reduce the effects of overestimation [Atiya and Hager, 1991].

In the mapping system developed here, a design decision was made to use regular intervals, as described in Chapter 5. A full discussion of interval arithmetic can be found in Appendix A.

4.5 Abstracting Behavior

When we introduced the path graph in Section 4.3 above, we said that arcs between waypoint nodes are labeled by the actions which move the robot between the corresponding waypoints. We thus implicitly assumed that actions could be treated as atomic labels. However, traveling between waypoints in general requires many primitive actions coordinated with one another: following hallways, acquiring designators, aligning with waypoints, and so on. We must therefore address the problem of how to represent this complex behavior, and how to extract this representation from information available to the mapper. This is non-trivial because we cannot assume that robot behavior is chosen specifically to go between waypoints. If it were, we would presumably have ready-made (by the deliberator) plans to go from waypoint to waypoint which could be used directly. As it is, we really only know when waypoints are encountered during execution of some plan; the plan may or may not be reducible to a series of independent waypoint-to-waypoint subplans.

We must be explicit about our goals in representing actions. We do not aim to reason about the effects of the actions we represent (except to say that an action sequence gets us from waypoint A to waypoint B). Instead, our goals are simpler: *behavior repetition* and *behavior recognition*. Behavior repetition is important when planning, so that a plan consisting of a sequence of action arcs in the path graph may be performed by executing, in sequence, plans associated with the arcs. Repetition should be reliable, but it need not be perfect, provided failure can be detected (since we assume reactive execution). For mapping, we also require representations that support behavior recognition, so that the mapping system can tell when a known action arc has been traversed. A small proportion of false negatives (not recognizing execution of a known action) is permissible, since the main result of a false negative is to add an extra action description to the map.

In the remainder of this section, we will develop a behavior representation scheme which addresses these goals. We first formulate the problem by specifying the input in terms of primitive actions and the output as robot plans. We then develop a simple method for translating sequences of executed primitive actions to plans which repeat the observed behavior. We then illustrate the application of the method to several examples in the Ars Magna simulator. This section closes with a discussion of the limitations of our translation technique, and a sketch of some proposed knowledge-based translation methods to overcome these problems.

4.5.1 Primitive actions

What can the robot's mapping system know about its behavior? Since our passive mapping paradigm introduces a modular separation between mapping and action selection, we need to be explicit about what sorts of behavior-describing inputs are supplied to the mapper. In simple domains such as the classical blocks world, actions are atomic, and may be represented by simple formulas such as `puton(A,B)`. Continuous behavior can also be easily represented when it is homogeneous—if it can be expressed algebraically, for example. However, nonhomogeneous, continuous behavior is more difficult to deal with. In particular, this is true for autonomous mobile robots—actions are not atomic, multiple actions can occur together, and trajectories are not smooth or easily described mathematically. Our task is to extract some useful description of this complex behavior (supporting repetition and recognition) by observing it.

The simplest way to represent the robot's behavior is to note its trajectory through space during the time period of interest. This approach has some fundamental difficulties, however. First is sensor and effector noise—the representation will not match the original trajectory, and future repetition of the same behavior will not result in the same trajectory. Moreover, if the robot does not start in exactly the same configuration, when executing the remembered trajectory, it may not reach its proper destination (or even crash into a wall). This problem is caused by the fact that repeating a specific trajectory is, by its very nature, a sort of open-loop control[5]. If the world is not entirely static, this problem is compounded, since the desired trajectory

[5] At a high-level, that is. Feedback control will be required, of course, in order to accurately follow the reference trajectory.

may become obstructed. Our solution is to represent behavior at a higher level of abstraction than geometric motion, by representing significant events in the robot's reactive control system (as described in Section 2.4.1). Our formulation also allows the expression of sensory feedback in the action representation.

Effector events

The primitive of our action representation is the *effector event*. Recall from Chapter 2 that low-level control is implemented by effector modules which perform tasks such as wall-following or object-approach. A robot action, in this paradigm, is a period of activation of one of these modules. The simplest representation of behavior, then, is as a sequence of effector module invocations/completions. More generally, though, effector modules provide us with a stream of signals signifying important *effector events*. Such events include invocation (possibly with some parameters) and completion (possibly with an error code and return values), as well as other, module-specific events (eg, an approacher may signal when the robot deviates from a straight path). Effector events constitute important transitions in the robot's behavior, either in terms of activation/deactivation of different effector modules, or in terms of changes within a particular module. This indicates that a sequence of effector events (with associated parameters and signals) is a reasonable representation of robot behavior. Its utility for repetition and recognition is an empirical question, which is addressed later on in this section.

Events are represented by *event specifications*, which are messages sent from the relevant effector modules to the mapping system. An event specification consists of the name of the effector module, the type of event (eg, activation or error codes), the robot resources required (such as a camera or an arm), and the values of any parameters or signaled values. For example, the event specifier:

```
((Camera-Get-Place-Designator :ON) (:Camera 1) Door)
```

denotes the event of Camera-Get-Place-Designator being activated to find a designator for a doorway using Camera-1. The denouement of this action comes with when the module completes its work, signaling the designator it found, with:

```
(((Robot-Align-Place :ON) (:Robot))
 ((Robot-Align-Place :OFF) NIL (:Robot) 0.05)
 ((Robot-Go-Forward :ON) (:Robot) 1)
 ((Robot-Go-Forward :OFF) NIL (:Robot))
 ((Camera-Get-Place-Designator :ON) (:Camera 0) (Door))
 ((Camera-Get-Place-Designator :OFF) NIL (:Camera 0)
  (:Designator 43 0 0.3 :Non-Local (Door)))
 ((Camera-Track :ON) (:Camera 0)
  (:Designator 43 0 0.3 :Non-Local (Door)))
 ((Robot-Goto-Designator :ON) (:Robot)
  (:Designator 43 0 0.3 :Non-Local (Door)))
 ((Robot-Goto-Designator :OFF) NIL (:Robot)
  (:Designator 43 0 0.3 :Non-Local (Door)))
 ((Camera-Track :OFF) :LOST-TRACK (:Camera 0)
  (:Designator 43 0 0.3 :Non-Local (Door)))
 ((Robot-Align-Place :ON) (:Robot))
 ((Camera-Get-Place-Designator :ON) (:Camera 0))
 ((Robot-Align-Place :OFF) NIL (:Robot) -0.05)
 ((Camera-Get-Place-Designator :OFF) :DESIGNATOR-NOT-FOUND (:Camera 0))
 ((Robot-About-Face :ON) (:Robot))
 ((Robot-About-Face :OFF) NIL (:Robot) 3.11)
 ((Robot-Go-Forward :ON) (:Robot) 1)
 ((Robot-Go-Forward :OFF) NIL (:Robot)))
```

Figure 4.5: An event trace. The robot is going from a doorway to one opposite it.

```
((Camera-Get-Place-Designator :OFF) NIL (:Camera 1)
 (:Designator 643 1 0.4 :Non-Local (Door)))
```

The `NIL` indicates no error code (i.e, success). The designator components are, in order, a unique identifier, the camera used to find the designator, its relative direction, its type (local or non-local), and its waypoint type. Since we are interested here only with navigational behavior, we will only consider events on effector modules related to robot motion or relevant perception tasks; other kinds of events, such as manipulations, are ignored. We term a list of event specifications representing robot behavior over an interval of time an *event trace*. Figure 4.5 gives an example of a simple event trace.

4.5.2 The RPL Plan Language

We just discussed how behavior may be represented as effector event traces. Our objective is not only to describe the robot's behavior between waypoints, but to represent it in such a way that the robot can construct plans for its high-level goals (such as delivering packages) using the behavioral information stored in its map. This is the goal of *behavioral repetition*. To do this, we must translate from event traces to some kind of executable plan notation. The plan notation used here is McDermott's RPL, a reactive plan language developed for work in transformational planning [McDermott, 1991]. RPL is a parallel programming language with special constructs for reactive plan execution. This section will describe those RPL features used in this dissertation[6]. While some of the features described here are not used in this chapter (for event translation), they are made use of in later chapters, for exploration and goal achievement.

At the most basic level in RPL are forms for control flow. Commands may be executed in sequence by `(seq` a_1 a_2 ... a_n`)`, and in parallel by `(par` a_1 a_2 ... a_n`)`. Conditionals are expressed (as in Lisp) by `if` (though the condition is treated specially if it contains fluents, see below). More interesting is the way subplan *failure* affects control flow. In order that plans may react intelligently to conditions in the world, RPL contains explicit constructs for expressing plan failure. The statement `(fail)` causes the plan executing it to fail. It may be invoked because of an effector failure (signaled by an effector module), or because a plan decides that it cannot succeed (eg, a door-opening plan which determines the door is locked). A subplan failure causes its supertask to fail as well, unless the failure is caught by a failure-handling construct. For example, `(try-all` a_1 a_2 ... a_n`)` tries all its subplans, and succeeds iff any subplan does. This allows a plan to contain multiple methods for achieving a single task; if some fail, others may succeed. There are a number of RPL constructs for dealing with subplan failure; those we use will be described when necessary.

Since multiple processes may be active at any given time, RPL provides two synchronization mechanisms. The first is the *valve*, which is a kind of semaphore. Only a single process may own a valve at a given time (subplans inherit from their parent plans). If a process requests a valve (via `valve-request`) which is owned by another

[6] This section does not purport to contain a complete description of the RPL language. Please refer to [McDermott, 1991] for a complete description and discussion of RPL features.

process (not an ancestor), the process is *blocked* and cannot run until the valve is released (by `valve-release`). Valves are used to control contention over robot resources, such as the wheels. Another synchronization mechanism, which is also used for interprocess communication, is the *fluent*. Fluents represent time-varying values, or channels of communication. Fluents are created by `create-fluent`. The value of a fluent may be set or read by any RPL process via `set-value` and `fluent-value`. Effector module signals are also implemented in terms of fluents set by the modules. Compound fluents may be constructed through the overloaded use of functions such as `and` or `+`; the result returned is a fluent which is constrained appropriately by the values of the argument fluents. Control constructs such as `if` automatically 'dereference' fluents in condition tests, so, if f is a fluent or a fluent-valued expression, `(if` f `(a) (b))` is equivalent to `(if (fluent-value` f`) (a) (b))`. Processes may also *wait* for a fluent to become non-NIL or to change value, by `(wait-for` f`)` and `(wait-for (changed fluent` f`))`. There are also some more sophisticated ways to use fluents to control program execution. One example is the use of `(filter` f a`)` which executes a just as long as f is true, terminating when either a completes or f becomes false. Another useful construct is `(whenever` f a`)` which executes a whenever f becomes true.

Often a plan may be executed in order to constrain other plans. McDermott [1978] terms such plans *policies*. For example, a plan may contain a policy that monitors the robot's batteries and recharges them when they run low. Policies can also implement run-time protections; for example, if a robot drops the object it is carrying, it should stop and pick it up. Policies are expressed in RPL by `(with-policy` p b`)`, where p is a policy which constrains b. This means that whenever p runs, b is blocked (so that p can fix things before b continues). Usually, p will be a plan that never terminates (except possibly in failure) and is usually blocked, such as the common locution `(whenever` *problem repair*`)`. If a policy fails, the entire `with-policy` plan fails, since the constraint has been irreparably violated. If a policy succeeds, then the constraint is deemed moot, and goes away.

Nisp notation

RPL is written in Nisp [McDermott, 1988], a portable Lisp macro package with compile-time typing. Nisp code looks mostly like Common Lisp code, with some exceptions. As far as we are concerned here, the relevant differences are[7]:

- Variables may be declared when bound. A type is declared by putting a hyphen followed by a type designator after a variable name. Type designators are either type names, or compound type descriptions, such as `(Lst` t`)`, denoting 'list of objects of type t'. An object of type `(~` t`)` is either `#F` or of type t.
- If x is a structure of type t, `(!_(`t s`)` x`)` accesses x's s slot.
- The macro `!=` is essentially a synonym of `setf`.
- Boolean values are given as `#T` and `#F`, for clarity.

4.5.3 Translating event traces into RPL

As discussed above, the robot's behavior during an interval of time is given by the effector module activations in that interval. This behavior structure may be represented by an *event trace*, which is a sequence of event specifications. In order to represent this behavior so that it can be repeated, we need to translate event traces into RPL programs which can then be executed and the behavior repeated.

Rule-based translation

Since effector modules are basically stateless, effector events are nearly context independent. This means that we can translate an event trace into a RPL plan by translating each event in the trace and then gluing them together. Individual event specifications are translated by a set of translation rules, whose left-hand-sides are event specifications with variables to be instantiated. The right-hand-side of each rule is a plan fragment with variables to be substituted for.

[7] This summary is adapted from that in [McDermott, 1991].

The plans we will construct are sequences consisting of three sorts of steps: effector activations, variable assignments, and condition tests. Activations provide parameters to effector modules, while signal and finish events supply both values the plan can later use and conditions that must be checked for the plan to continue (such as error codes).

Activation events become plan steps which activate effector modules and supply needed parameters. For example, we have a translation rule for the event of aligning a camera with the direction of the robot:

```
(def-trace-step-rule ((Camera-Align-Robot :ON) (:Camera ?c))
  (set-value (vref camera-error* !@c) NIL)
  (rpl-camera-align-robot !@c) )
```

The first argument of the form is the event specifier to be matched; `?c` is a variable which will be bound to the index of a camera. The remaining forms constitute the RPL form, inside an implicit `seq`. Here, `!@`x indicates that the value of variable x should be substituted in. This rule operates to translate an event such as:

```
((Camera-Align-Robot :ON) (:Camera 1))
```

into the effector-invoking RPL plan:

```
(seq (set-value (vref camera-error* 1) NIL)
     (rpl-camera-align-robot 1) )
```

The fluent which represents the module's error code is first set to `NIL` so that the plan can tell if it is set by the module. Most invocation translation rules are as simple as this one, since all they must do is supply the needed parameters to an effector module.

Condition handling plan steps wait for an effector signal, check error codes for failure, and ensure that the values returned are those expected. So, for example, the event above might eventually result in:

```
((Camera-Align-Robot :OFF) NIL (:Camera 1) 3.13)
```

This corresponds to a plan step ensuring that the camera has completed its motion and that no error occurred[8]:

```
(seq (wait-for (vref camera-moved* 1))
     (if (not (null (vref camera-error* 1)))
         (fail)) )
```

This plan fragment waits for the camera to move and then fails if an error occurred. More information can be added to the failure descriptor to provide more information for failure recovery. The general translation rule for this case is:

```
(def-trace-step-rule ((Camera-Align-Robot :OFF) ?err
                      (:Camera ?c) ?d-ang)
  (wait-for (vref camera-moved* !@c))
  (if (not (eq (vref camera-error* !@c) !@err))
      (fail)) )
```

Note that we check that the error code at execution is the same as that in the original trace; effector failure does not always mean plan failure. More complicated condition tests might be used to make run-time errors result in recognizable subplan failures.

Testing error codes is relatively simple. When values are signaled, however, more needs to be done. The signaled value must be bound to a variable and should be tested for conformance to the event specification. For example, the event specification:

```
((Camera-Get-Place-Designator :OFF) NIL (:Camera 1)
 (:Designator 43 1 0.4 :Non-Local (Door)))
```

describes the successful acquisition of a non-local place designator on a doorway. Designators are uniquely labeled with identifiers, in this case 43. The plan fragment generated from the event should ensure that the designator returned conforms to this specification. More concretely, the proper plan fragment is:

[8] 3.13 represents the amount the camera actually turned (in radians). It is not used here.

```
(seq (wait-for (vref camera-input* 1))
     (if (not (eq (vref camera-error* 1) ())) (fail))
     (when (null (fluent-value (vref camera-error* 1)))
       (!= desig-1 (car (fluent-value (vref camera-designators* 1))))
       (if (not (and (eq (!_ (DESIGNATOR type) desig-1) :Non-Local)
                     (equal (!_ (DESIGNATOR data) desig-1) '(Door))))
           (fail))))
```

This plan sets a variable to the effector's signal and checks its slots' values to see if they conform to those in the specification. The rule that produces this is:

```
(def-trace-step-rule ((Camera-Get-Place-Designator :OFF) ?err (:Camera ?c)
                      (:Designator ?id ?d ?ang ?d-type ?wtype))
  (wait-for (vref camera-input* !@c))
  (if (not (eq (vref camera-error* !@c) !@err))
      (fail))
  (when (null (fluent-value (vref camera-error* !@c)))
    (!= !@(plan-designator-var id)
         (car (fluent-value (vref camera-designators* !@c))))
    (if (not (and (eq (!_(DESIGNATOR type) !@(plan-designator-var id))
                      !@d-type)
                  (equal (!_(DESIGNATOR data) !@(plan-designator-var id))
                         '!@wtype)))
        (fail)) ))
```

Each designator has a unique identifier (an integer) which is associated at translation time with a plan variable by `!@(plan-designator-var id)`. The plan variables thus created are bound in the scope of the entire trace plan. These variables allow signals from one effector module to be used by another. Thus, the above example of a designator acquisition may appear in a trace before the event:

```
((Camera-Track-Designator :ON) (:Camera 0)
 (:Designator 43 0 0.4 :Non-Local (Door)))
```

This event denotes the beginning of the tracking of the acquired designator, so the proper variable is inserted into the plan:

```
(seq (if (null desig-1) (fail))
     (if (and (eq (!_(DESIGNATOR type) desig-1) :Non-Local)
              (eq (car (!_(DESIGNATOR data) desig-1)) 'Door))
         (seq (par (set-value (vref camera-error* 0) NIL)
                   (set-value (vref camera-tracking-error* 0) NIL))
              (rpl-camera-track 0 desig-1)
              (wait-for (or (vref camera-tracking* 0)
                            (vref camera-tracking-error* 0)))
              (if (not (vref camera-tracking* 0)) (fail)))))
```

This plan fragment checks the designator to be tracked, starts the tracker, and then waits for the camera to either acquire the track or fail.

Variables and codesignation

Use of plan variables allows plan steps to depend on one another by identifying values produced and used at different points in the trace. Determining which signaled designators are identical is simple, since they cannot be modified and each can be assigned a unique identifier. However, value identification is impossible in general for other types of values, such as numbers, for which these properties don't hold. If '3' appears as a parameter after a '5' is signaled in an event trace, we don't know if the '3' is a separate constant, or $x-2$, with x set to '5' in this trace, or any other of an infinite set of possibilities. Since the identification task is impossible without more knowledge, we finesse the issue, and use the heuristic assumption that equal values are actually the same. Since it is unlikely that two randomly chosen real values will be equal, it is probably safe to assume that signals that equal subsequent parameters refer to the same logical quantity. Quantities that are unequal are assumed to be unrelated; while this might be false, the knowledge needed to resolve this identification problem is lacking here.

Post facto parameterization

Besides codesignation, there is another sort of dependence between steps in an event trace. It arises from the fact that knowing the best way to reproduce a robot action may depend on knowing its results, in addition to its parameters. For example, a command to turn the robot 90° may actually turn the robot 94.6°. A subsequent command to turn the robot 90° may end up turning the robot 85.2°, introducing a repetition error that is larger than strictly necessary. To avoid this problem, the original "turn 90°" should be translated as[9] "turn 94.6°". This phenomenon is not exclusive to open-loop actions. It also arises in perceptual routines, where this sort of *post facto parameterization* is needed. For example, when the robot looks for a doorway and two are in its field-of-view, the direction of the designator which is found should be inserted into the perceptual command when the robot attempts to find the doorway again, so that the correct doorway is found.

The solution to this problem is to have translation rules look for other relevant events in the trace to instantiate needed parameters. So, for example, the activation of turning left by $\pi/2$ radians:

```
((Robot-Turn-Left :ON) (:Robot))
```

may actually get translated into the plan:

```
(seq (set-value robot-movement-error* NIL)
     (rpl-robot-turn 1.4))
```

provided that the matching completion event signals that the robot actually turned 1.4 radians, as by, for example, `((Robot-Turn-Left :OFF) NIL (:Robot) 1.4)` . This makes it more likely that repetition will closely approximate the original behavior. Post facto parameterization is implemented by allowing translation rules to look at the context of the rest of the trace, as in the rule for this example:

```
(def-trace-step-rule ((Robot-Turn-Left :ON) (:Robot))
  (:CONTEXT (after (first ((Robot-Turn-Left :OFF) ?err
                           (:Robot) ?ang))))
  (set-value robot-movement-error* NIL)
  (rpl-robot-turn !@ang))
```

[9] If, however, errors are partially systematic, the problem of reproduction is not as severe; hence an average of the original command parameter and the result may be used instead.

The `CONTEXT` clause here specifies that if the robot completed turning and signaled an actual turn of `ang` (regardless of any error code produced), then the robot should try to turn by that amount.

Preprocessing

The core of the behavior translation system is the rule-based translation scheme described above. In addition there is some preprocessing to event traces that is needed to ensure that the plans produced are robust. There are two sorts of preprocessing described here: *elision* of unnecessary effector events, and *prefixing* of needed event activations. They are described below.

Event elision

One small complication is that some trace steps reflect actions which do not affect the results of the plan in any way. This can be a problem if the results of these actions are not reliable. In such cases, requiring such actions to be performed (and their results to be consistent with the original trace) will sometimes cause the resultant plan to fail unnecessarily. The solution is to elide those steps from the event trace before translation, so that they can't affect the results. The trick is detecting such irrelevant steps. Most actions, in fact, are relevant in some way; any action that moves the robot, for example, will affect the results of its plan. However, perceptual events may not be needed, and may be variable enough to require elision. For example, if the robot acquires a designator that is never used, the acquisition event should be ignored, since designator acquisition is not entirely reliable.

Prefixing

A more serious problem requiring trace preprocessing is *dangling activation*. Consider the case of an event trace meant to represent the robot's behavior between two corridor intersections (as in Figure 4.6). If the robot did not stop at the first intersection, but simply continued to follow the wall, activation of the wall-following behavior will not be reflected in the trace produced by observing effector events only between the

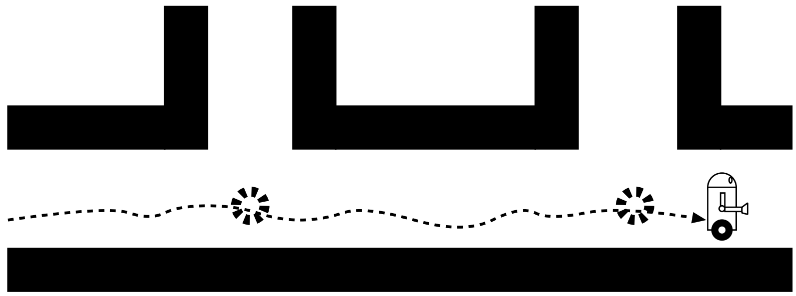


Figure 4.6: Dangling activation for a robot following a wall down a hallway. The stars mark waypoints.

intersections. Therefore, simply translating the trace will result in a plan that does not move the robot at all! Dangling activation can be dealt with by adding a virtual activation event for the dangling effector module at the beginning of the trace. This works because effector modules are essentially stateless, so activating the module at an intermediate point will produce the same future behavior. Another problem that is solved by prefixing virtual events to traces is when an effector activation depends on a designator whose acquisition is not within the trace. This is solved by prefixing virtual designator acquisition events for the required designators.

Both types of prefixing require taking a snapshot of aspects of the effector subsystem state when trace recording starts—which modules are active (and their parameters), for dangling activation, and the state of current designators, for virtual designator acquisition. Currently, this is not fully implemented. A simpler form of activation prefixing is used, which examines the trace for end events without matching activation events, and prefixes appropriate activation events to the trace. This only works if the dangling effector module completes within the trace; this happens to be the case in our plans. If this is not the case, the trace-gathering mechanism must be made more sophisticated, and query the reactive control system for the states of active modules when trace-gathering begins. A similar method is used by Torrance [1994] to deal with dangling activation. His execution system and plan representation are much simpler than ours, however, easing the task considerably.

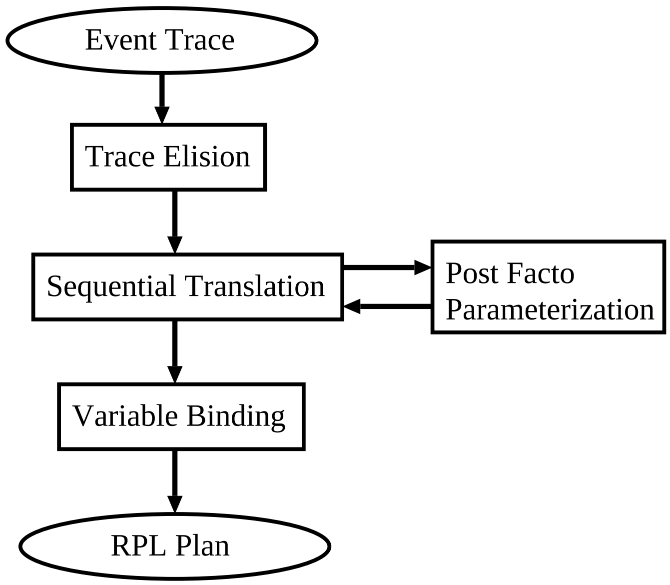


Figure 4.7: Schematic of the trace translation algorithm.

4.5.4 Trace translation

Event trace translation proceeds as depicted in Figure 4.7. First, irrelevant trace steps are elided. Then, the steps in an event trace are translated sequentially by applicable translation rules. When a post facto parameterizable step is encountered, the rest of the trace is searched for the needed parameter, which is inserted in the translation. If an event signals a value, a new variable name is created, which is indexed under the value in a variable table. Then, when a value is used as a parameter, the value's variable name is inserted, if it exists, otherwise the value is assumed to be a hard-wired constant. After all the steps are translated, they are wrapped in a `let` form binding the local variables in the plan. This implicitly executes the plan steps in the same order that they appear in the trace. Applying this sequential translation method fully to the trace in Figure 4.5 above gives the plan in Figures 4.8 and 4.9.

```
(let ((desig-1 NIL))
  (seq (seq (set-value robot-place-input* NIL)
            (rpl-robot-place)
            (wait-for robot-place-input*))
       (if (not (null robot-place*))
           (seq (set-value robot-movement-error* NIL)
                (rpl-robot-align-place))
           (seq (set-value robot-movement-error* NIL)
                (rpl-robot-turn 0.05))))
  (seq (wait-for robot-stopped*)
       (if (not (eq (fluent-value robot-movement-error*) NIL))
           (fail)))
  (seq (set-value robot-movement-error* NIL)
       (rpl-robot-go-forward 1))
  (seq (wait-for robot-stopped*)
       (if (not (eq (fluent-value robot-movement-error*) NIL))
           (fail)))
  (seq (wait-for (vref camera-input* 0))
       (if (not (eq (vref camera-error* 0) NIL))
           (fail))
       (when (and (null (fluent-value (vref camera-error* 0)))
                  (not (null (fluent-value (vref camera-designators* 0)))))
         (!= desig-1 (car (fluent-value (vref camera-designators* 0))))
         (seq (set-value (vref camera-moved* 0) NIL)
              (rpl-camera-turn 0 (- 0.3))
              (wait-for (vref camera-moved* 0)))
         (if (not (and (eq (!_(DESIGNATOR type) desig-1) :Non-Local)
                       (equal (!_(DESIGNATOR data) desig-1)
                              '(Door))))
             (fail))))
```

Figure 4.8: First part of sequential plan generated automatically from the event trace in Figure 4.5. Note the length and complexity of the plan generated from a relatively short piece of behavior.

```
(seq (if (null desig-1) (fail))
     (if (and (eq (!_(DESIGNATOR type) desig-1) :Non-Local)
              (eq (car (!_(DESIGNATOR data) desig-1)) 'Door))
         (seq (par (set-value (vref camera-error* 0) NIL)
                   (set-value (vref camera-tracking-error* 0) NIL))
              (rpl-camera-track 0 desig-1)
              (wait-for (or (vref camera-tracking* 0)
                            (vref camera-tracking-error* 0)))
              (if (not (vref camera-tracking* 0))
                  (fail)))))
(seq (wait-with-timeout (vref camera-tracking-error* 0) 30)
     (if (not (eq (vref camera-tracking-error* 0) :LOST-TRACK))
         (fail)))
(seq (seq (set-value robot-place-input* NIL)
          (rpl-robot-place)
          (wait-for robot-place-input*))
     (if (not (null robot-place*))
         (seq (set-value robot-movement-error* NIL)
              (rpl-robot-align-place))
         (seq (set-value robot-movement-error* NIL)
              (rpl-robot-turn -0.05))))
(seq (wait-for robot-stopped*)
     (if (not (eq (fluent-value robot-movement-error*) NIL))
         (fail)))
(seq (set-value robot-movement-error* NIL)
     (rpl-robot-about-face))
(seq (wait-for robot-stopped*)
     (if (not (and (eq (fluent-value robot-movement-error*) NIL)
                   (<= (angle-dist (fluent-value robot-angle-diff*) 3.11)
                       0.2)))
         (fail)))
(seq (set-value robot-movement-error* NIL)
     (rpl-robot-go-forward 1))
(seq (wait-for robot-stopped*)
     (if (not (eq (fluent-value robot-movement-error*) NIL))
         (fail))))
```

Figure 4.9: Continuation (from Figure 4.8) of the sequential plan generated automatically from the event trace in Figure 4.5.

4.5.5 Recognition

RPL notation is a needless complication for trace recognition. The *recognizable* version of a trace is a preprocessed trace which has been post facto parameterized. Such a trace represents the original behavior modulo the considerations discussed above of robustness and generality. When comparing two recognizable traces, matches between designators are done based on their descriptions, with multiple occurrences of a designator in one trace constrained to match to the same designator in the other. Numerical parameters are matched to within a threshold. Different thresholds are given for different types of trace steps; the accuracy required for turning (0.1 radians) differs from that required for forward movement (0.3 meters).

4.5.6 Examples

In this section we will examine the performance of the trace translation technique described above on several examples, using Ars Magna. The examples demonstrate the strength of the method, particularly the use of effector stability to produce stable behavior repetition. Recall that the purpose of trace translation is to provide a way of labeling path graph links with plans which move the robot between waypoints. The examples here illustrate the robot moving among several waypoints at a go; repeating a behavior just between two waypoints is a simpler task. The evidence thus suggests that trace translation will provide links plans which are robust enough to support navigational planning.

Simple example

The first example demonstrates the stable repetition of a short piece of robot behavior. The robot behavior was generated by hand; the experimenter manually activated effector modules in sequence. Figure 4.10 shows the original trajectory of the robot in a simple world with no obstacles. The event trace generated by this behavior contained 48 event specifications. The translated RPL plan was run ten times in each of two situations. The first situation was identical to the original one. These runs are summarized in Figure 4.11(a), which depicts the robot's trajectories on

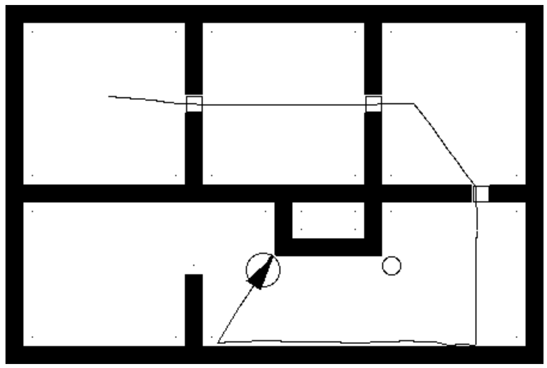


Figure 4.10: Robot trajectories for a hand-generated behavior. The robot was started in the center of the first room. (Circles and dots denote waypoint types.)

those ten runs. Note the variance in the precise trajectories the robot followed, due to motion noise. The plan failed in one of these runs, when the robot lost track of the final corner on its approach due to perceptual noise. The plan was also run in the same world with the addition of some obstacles (Figure 4.11(b)). As is clear from the figure, the existence of a low-level obstacle avoidance behavior enabled the robot to stably repeat its original behavior quite well. Note that different trajectories were followed on different runs, depending on how the robot decided (locally) to go around the obstacles. One of these ten runs also failed, when the robot lost track of the third door in its trajectory due to perceptual noise. Note that both plan failures were of the sort that cannot be avoided without more detailed domain knowledge; they do not represent failures of the plan translation system.

Plan-generated examples

The next two examples are of behaviors generated by a randomized wandering plan. Figure 4.12 shows the robot's trajectories during execution of the initial behavior (a) and the test plan (b) for the first example. The robot started in the rightmost doorway facing down, and ended up, as shown, in a corner of the rightmost upper room. The event trace contained 72 event specifications. Despite the variance in the robot's trajectory out of the door, 9 of the 10 tests succeeded. The failure occurred

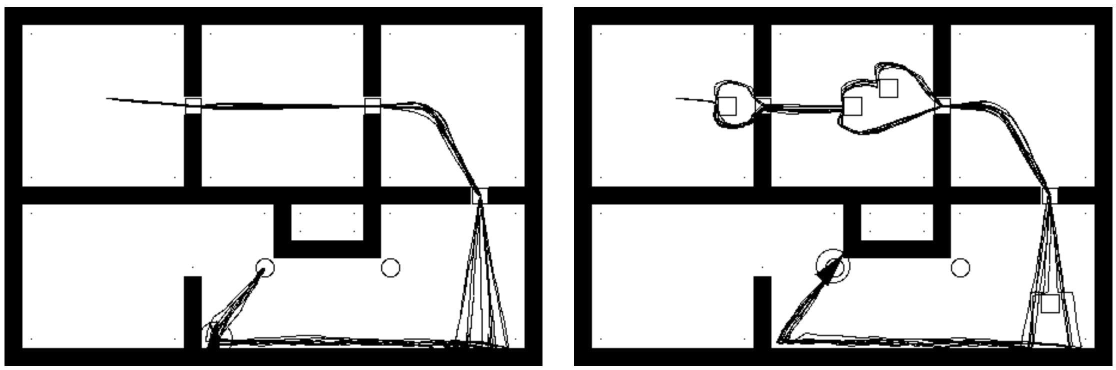
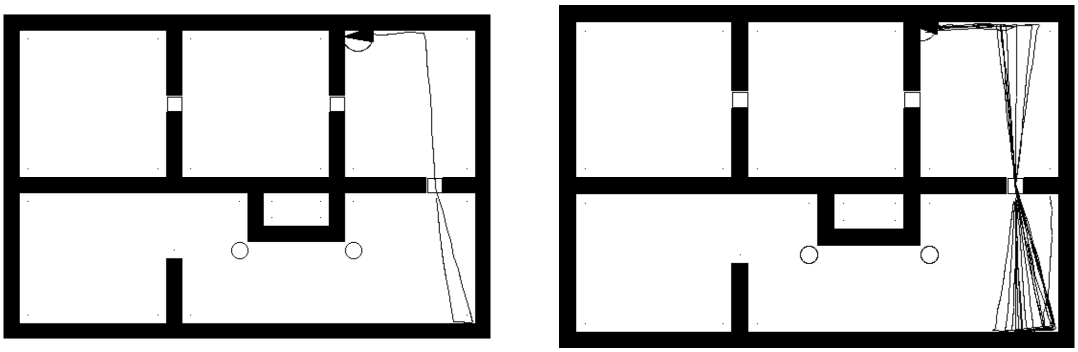


Figure 4.11: Behavior repetition tests for the example in Figure 4.10. (a) Composite of trajectories of 10 executions of the translated plan. (b) Composite of trajectories of 10 tests with obstacles (squares) added.

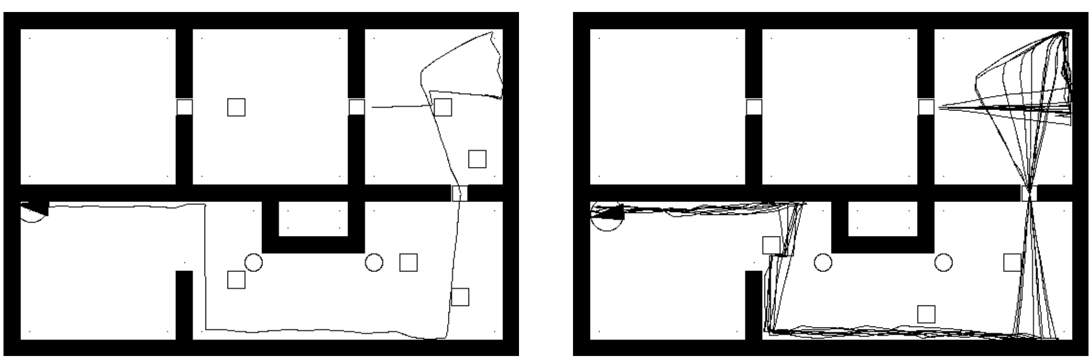
when the robot failed to acquire a designator for the door on its return; it ended up in the upper-right corner of the lower room, as shown.

The second example of automatically-generated behavior is depicted in Figure 4.13(a). The robot was started in the second doorway from the left, facing right. While describing a longer and more complex behavior than the last example, the event trace generated here was shorter, with 60 events in all. The behavior in this example was also somewhat more fragile than those in previous examples, because it depends on the robot heading for the door it started from and then losing track of it due to occlusion. (This happens at the kink in the robot's trajectory, where it begins heading for the lower doorway.) Ten test runs were run on this example, with obstacles in different places than when the behavior was originally generated. Three of the test runs failed because the robot headed for the lower doorway first. In all the other runs, the robot attained its goal, despite the different positions of the obstacles. Also note that the trajectories followed by the robot on different trials differed greatly, though the overall effect was the same. This illustrates the problems of using trajectories to represent behavior (as discussed above).



(a) (b)

Figure 4.12: Robot trajectories for a short example of translation of a plan generated event trace. The robot was started in the rightmost doorway facing down. (a) Initial trajectory generated by wandering. (b) Composite of trajectories of 10 test runs.



(a) (b)

Figure 4.13: Robot trajectories for one example of translation of a plan generated event trace. The robot was started in the center of the second room. (a) Initial trajectory generated by wandering. (b) Composite of trajectories of 10 test runs.

4.5.7 Limitations and extensions

While the trace translation algorithm described above works well in many cases (as illustrated by the examples above), it has a number of important limitations. Some can be remedied by simple extensions to the algorithm, while others are inherent in our whole approach. In this section, we will discuss the main areas in which our method has difficulties, and also sketch some extensions and changes to the approach that should remedy these shortcomings. None of these are currently implemented.

Timing

The most obvious limitation of our technique is the fact that the plans it produces have no notion of time; they are purely reactive. If two events A and B are adjacent in a trace, the translated plan will execute B right after A, even though there may have been a period of time between A's completion and B's start in the original behavior. This can cause problems when repeating B if the situation changed after A completed, such as if the robot was moving. We propose two possible solutions. One is to incorporate a timing "effector" module into the control system, so that when the deliberator wishes to explicitly wait for a period of time, it activates the timing module with the desired wait time, which module then signals when the time is up and the plan should continue. These events go into the event trace like any other and become a part of the translated plan, causing the robot to wait appropriately upon repetition. This only works, of course, if the original wait is explicit; if the wait is caused implicitly, say by the deliberator's cogitation, another approach is required. Here, we note that the main difficulty is caused if the robot is moving between events A and B, so that it is at the wrong place for B if it doesn't wait. This can be solved by using odometry to measure the approximate distance moved between adjacent trace events. If the distance moved between a signal event and the activation event following it is significant, the plan produced will wait after the signal before performing the activation until the robot has moved approximately the requisite amount. Both these methods are useful, the former for those cases where the robot must wait for some external event (which cannot be directly sensed) to occur, the latter for when the robot is thinking as it moves.

Effector failure

The main reason that translated plans failed in our experiments (on those few occasions that they did) was due to effector failure—losing track of a designator, getting stuck following a wall, and so on. Strictly speaking, this sort of failure is not a shortcoming of our trace translation technique, as these failures are beyond our control. However, there is a way to mitigate the effects of such runtime failures. A robust reactive plan for accomplishing some goal will usually contain quite a few failure recovery procedures, which handle cognizant failures of effector modules (see Section 2.4.1) to deal with run-time errors. Such recovery methods could be incorporated into translated plans by using domain-dependent heuristics. For example, if a trace contains a “go to designator” event, the translated plan may contain, in addition to an invocation of the “go to designator” module, a policy which attempts to recover the proper designator if it is lost before “go to designator” properly completes. Such heuristic policies could be developed for all common run-time failures, improving greatly the robustness of the resultant plans. Also, if trace plans are used as components in larger plans (as we propose for navigation), robustification of this sort will be applied during transformational planning.

Invalid constancy assumptions

Another difficulty, inherent in our trace translation approach, is caused by the fact that constants hardwired into the plan often shouldn’t be. These ‘invalid’ constants may be either explicit or implicit in the trace. An example of an explicit invalid constant would be a turning angle of 84°, when the robot was really turning to face halfway between two designators. There is no way to tell this from the event trace. Similarly, an implicit constant, such as the fact that the robot went into the third door on its left, may be invalid—the robot may really have gone into “the first open door on the left”; the fact that it was the third may be irrelevant. The knowledge needed to resolve the true roles of either implicit or explicit constants is simply not available from a bare event trace, and so pure trace translation will not suffice, no matter how elaborate we get. The proposal we sketch below uses the original plan which generated the trace as the basis for a more intelligent way of synthesizing plans to repeat robot behavior.

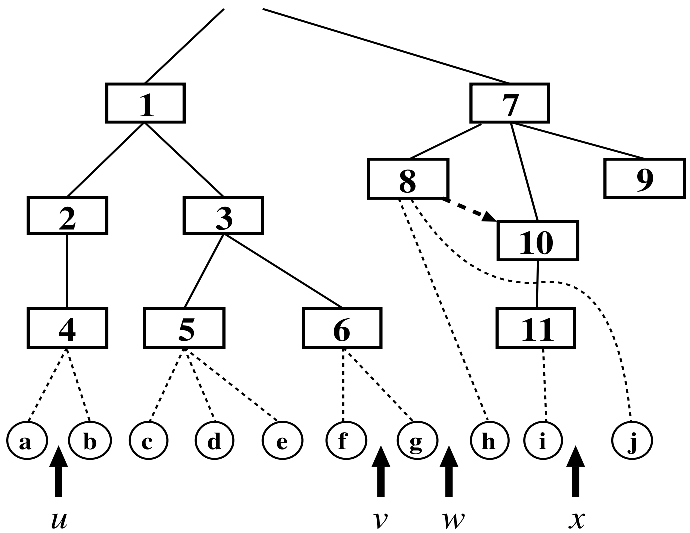


Figure 4.14: A schematic representation of a task net and the event trace it generates. Task 8 is a policy on task 10.

Task-net abstraction

When a RPL plan is executed, it generates a *task net* which is a graph whose nodes represent the instantiated portions of the plan (those which actually run). For example, the task net node for `(seq` A B C`)` will have children corresponding to the subtasks for A, B, and C. However, the node for `(if` X A B`)` will have only one subnode for either A or B, depending on which one actually got executed. Loops have one subtask for each iteration. Though it is not part of the RPL interpreter [McDermott, 1991], we can take the leaves of the task net to be effector event specifications. We now may use the task net to find the 'most general' plan that corresponds to the behavior represented in some event trace. The basic idea is to climb up the task net from the events we are interested in and find the highest tasks which are completely included in the scope of the trace. For example, Figure 4.14 shows a schematic representation of a task net (boxes) and its associated events (circles). Consider the trace between points u and v. The most general plan generating that trace is the plan associated with the sequence **b–5–f**, since task 5 is entirely within the scope of the trace. If we move the end of the trace we consider to w, we would then want the plan for **b–3**, since tasks 5 and 6 are the only subtasks of task 3. The plan associated with an event specification is its translation as above, while that for a task node is derived from the RPL code associated with the task (with parameters filled in from its instantiated subtasks). We call the complete plan derived for a trace by task-net abstraction the *abstracted plan* for the trace (contrasted with the translated plan as above). This method abstracts away from the particular behavior trace to the higher-level goals which generated it, in the process incorporating the relevant decision-making parts of the plan. We will sketch here a heuristic approach to such *task-net abstraction* which addresses the main issues involved.

Task-net abstraction helps with the problem of invalid constancy assumptions because the plan's structure gives us useful clues about the function of various values in the robot's behavior, allowing us to decide which are essential and should be fixed, and which may be allowed to vary. For example, take the case above where the robot decided to go between two designators it had acquired. At the trace level, there is no way to tell what the turn angle represents; however, if we abstract to the supertask, we get a plan fragment something like:

```
(robot-turn-to-angle (average (designator-angle desig-1)
                              (designator-angle desig-2)))
```

which encodes exactly the desired relationship between the robot's percepts and its behavior. Here, `desig-1` and `desig-2` are variables representing designators acquired earlier. This suggests a strategy similar to that used in explanation-based generalization [Mitchell *et al.*, 1986], in which constants are 'back-propagated' through a proof tree in order to find the weakest preconditions which allow the proof to remain valid. Those preconditions then describe a justified generalization of the example. In our case, free variables do not really exist, since we do not know what preconditions on variable values would guarantee us the same behavior[10]. Therefore, all variables unbound in the scope of the abstracted plan are fixed at their values in the original task net.

Some expressions may need to be fixed, like free variables; for example, an expression returning a random number for randomizing a choice should be fixed at its original

[10]That is, barring a detailed logical domain model, in which case some version of EBG might apply. Such a model would be prohibitive to construct, and almost certainly would not describe reality well. See Gat [1991] for a discussion of why rigorous 'theories' of behavior are largely useless in robotics.

value, so that behavior does not vary when repetition is attempted. Other expressions may also have to be fixed, because of scoping problems. In the above fragment, for example, if the designators in question were acquired before the beginning of the trace under consideration, references to them are meaningless within the scope of the trace. Hence we have no alternative but to treat the turning angle as a constant. Such scoping problems are obvious, though, so their effect on the robustness of the resulting plan can be estimated.

A similar sort of 'value fixing' is needed for plan step ordering. Even when the original plan was agnostic as to the order of two plan steps (say, through use of the `partial-order` construct), the abstracted plan should take into account the actual order in which events occurred, since it may be crucial for repeating that particular behavior. Hence, whenever all events generated from a task A precede those generated from task B, and the order of A and B is not constrained, we should add a constraint in the abstracted plan that A should precede B.

A final consideration relates to the use of policies in plans. As described above in Section 4.5.2, a policy is a plan which constrains another plan. We must therefore distinguish between those effector events which result from policies and those which do not. Specifically, if a trace includes events both from a policy and the task it constrains, the entire policy should be kept, as a constraint on the abstraction of the constrained events. This applies even if some events generated by the policy are not in the scope of the trace. For example, consider the trace between points u and x in Figure 4.14. The plan that is generated by task-net abstraction should include both task 10 (as its only event is **i**), and task 8 as a policy on it, even though task 8 also generated an event outside the scope of the trace. If a policy did not generate events within a trace, however, it may be considered probably irrelevant to the behavior represented by that trace.

Chapter 5

Passive Mapping

That we can feed this mind of ours
In a wise passiveness.
—William Wordsworth

Summary

In this chapter we will discuss the passive mapping system itself. We view the mapping system as a sort of *adaptive estimator* in which estimation is a way of explaining observations. In non-adaptive estimators, observations are explained by fitting them to a known model. In adaptive estimation, on the other hand, we recognize that sometimes the best explanation for an observation involves positing modeling errors. We embody this idea in the Adaptive State-space Architecture, which constitutes the framework for our mapping system. The core of the architecture is the 'matching/prediction' estimation loop. Predictions of the robot's current location from previous estimates and sensory data are matched to the map to produce new location estimates. As this estimation process proceeds, learning occurs as the map is adapted to observations. There are two types of adaptation: updating and restructuring. Updating is driven by the estimation process—when observations cannot be explained by assuming the map is complete and correct, the map must be changed to produce a consistent new location estimate. Single observations may not contain the information needed to correct some mapping errors, however,

so restructuring allows for map correction which integrates information from many observations over time.

In order to develop heuristic procedures for correcting mapping errors, we use a taxonomy of the types of errors that can occur. There are three main categories of mapping errors: *positional inconsistency*, *spurious elements*, and *structural anomaly*. Positional inconsistency is when either a waypoint or robot position estimate is incorrect. It occurs because of incorrect identification of the robot's location and odometric outliers. Positional inconsistency can be corrected by matching predictions to the map approximately; if no clear match possibilities exist, then positional inconsistency can be diagnosed and corrected. Spurious elements result when sensor or effector noise causes the robot to 'hallucinate' action links or waypoints. These are removed from the map if they are not consistently observed. Finally, structural anomaly is when either multiple nodes in the map represent a single real waypoint, or vice versa. These errors are corrected for by merging or splitting waypoint nodes. We present two methods for diagnosing structural anomalies, one based on statistical analysis of waypoint position estimates, and another based on analyzing waypoints' connections to their neighbors.

5.1 Mapping and Estimation

In previous chapters, we developed an abstract robot model including visual place recognition and a map representation strategy designed to support map learning for intelligent navigation. The subject of this chapter is the operation of the passive mapping system itself. Recall that the primary functions of a mapping system, in our view, are localization and navigation. Given this, from the standpoint of other cognitive modules which use the mapping system, it can be viewed as a kind of state estimator. Localization is just estimation of the robot's location (whether Cartesian or qualitative) from sensor readings and a world model. And navigation is based on (explicit or implicit) projection of the results of performing different actions; this is just hypothetical estimation. So, ignoring for the moment the fact that the mapper must learn its world model, the rest of the cognitive system can treat it simply as a location estimator (and generator of navigational plans). Of course, we want this estimator to learn its world model, so we need to consider *adaptive* estimation.

Adaptation can be brought into the estimation process by viewing estimation as a form of explanation. An estimation of a system's state based on sensor readings and a system model is, in a sense, an explanation of those sensor readings. If the model is assumed to be correct and complete, then all that is needed to explain the readings is to fit them into the model. How this is done depends on the nature of the model used. For example, for classical linear systems with Gaussian noise models, an incremental least-squares fit 'explains' observations [Gelb, 1974]. Robust estimators may allow 'bad' observations to be thrown out or weighted less, implicitly explaining them as outliers [Hampel *et al.*, 1986]. More generally, explanation of observation may also include hypothesized changes to the estimator's model. The best explanation for an observation, given a model, may be that the model itself needs changing. This view of a mapping system as an *adaptive estimator* dovetails neatly with our requirement of passive mapping. Provided that methods can be developed so that map errors can be discovered by explaining observations, without active experimentation, an adaptive estimator is a passive mapper as well.

5.1.1 The Adaptive State-space Architecture

These ideas are embodied, for estimation in discrete state-spaces (such as diktiometries), in the Adaptive State-space Architecture (ASA), depicted in Figure 5.1. Since the system's model of the world (or map) is to be modified, it is represented explicitly in the architecture. In our system this model is a diktiometry as described in Chapter 4. The core of the estimator is the matching–prediction loop. Prediction combines an estimate with system inputs in order to predict the new state of the system. Matching seeks to explain observations in the context of the prediction and the model. It is called 'matching' because for discrete state-space models estimation amounts to deciding which state the system has moved to. Since these state-spaces may be quite large, the estimator also includes an indexing module whose function is to find good candidate matches quickly from observations.

There are two sorts of adaptation possible. The first is *updating*, which changes the state-space model in accordance with estimation when explaining observations properly requires changing the model. For example, if observations do not accord with

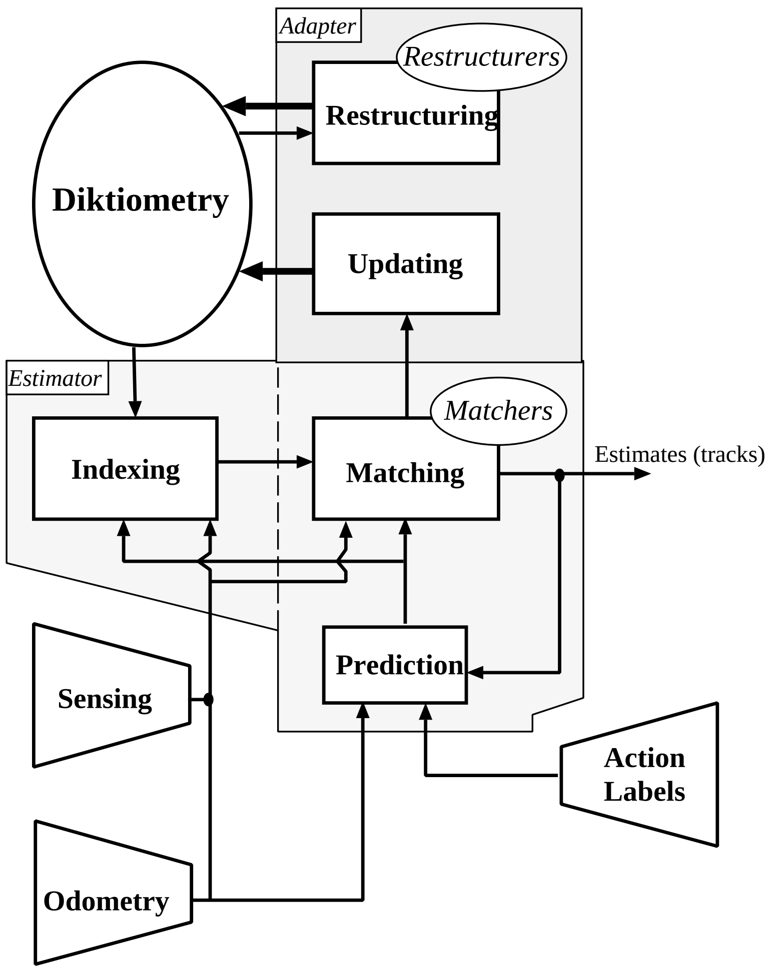


Figure 5.1: The Adaptive State-Space Architecture.

any known state, a new state should be postulated[1]. Not all necessary information for learning is present in single observations, however. It is often necessary to explain sets of observations in order to see what is going on. This will be the case if the space contains large-scale structure which is not visible from a single point. For example, consider an environment which contains two identical places A and B, each reachable from C via different doors. Unless the system can integrate information over time about the results of taking different doors from C, it may be unable to distinguish A and B. Doing this requires *restructuring*, which examines long-term observation statistics to correct mapping errors. The details of these various components of the ASA are explained in the rest of this chapter.

5.1.2 Tracks and odometry

We call estimates of the robot's location *tracks*, as they track the location of the robot as it moves. Multiple tracks may exist at any time, indicating multiple coexisting hypotheses as to the robot's location. A track t consists chiefly of three components: a position interval estimate[2] $\mathbf{x}_t$, an orientation interval estimate θ_t, and a waypoint node w_t. If multiple reference frames are used, we also have a reference frame F_t, with respect to which t's position estimates are given. In the rest of this chapter, it is assumed that transformations are made to equalize reference frames before comparisons or updates are applied. If t describes the location of the robot at its last waypoint encounter, it is said to be *current*. The actual position of the robot (assuming t correct) may be calculated by composing the track's estimate with odometry since the last waypoint encounter, $\mathbf{x}_o$ and θ_o.

[1]Unless the system is sure that its knowledge is complete. In such a case, an alternative explanation (say, outliers) may be preferred. This sort of context-sensitivity is discussed below in Section 5.5.

[2]Intervals are denoted generally by bold-face letters. An interval's endpoints are denoted as $\mathbf{x} = [\underline{x}, \overline{x}]$. The *interval hull* or *least bounding interval* $\Box S$ of a set S is the smallest interval completely including S. Appendix A contains more details about intervals and interval arithmetic.

5.1.3 Indexing

If we wish our robot to operate in an open ended fashion, our mapping system must be able to deal with large spaces containing thousands of waypoints. This can make it infeasible to examine all waypoints for a good match for robot localization. This section will describe some heuristic indexing methods for quickly finding likely match candidates. Each method is applied individually to each track t, giving a set of several candidate waypoints. The union of all the sets produced is used as input to matching to localize the robot (and subsequently update the map).

There are three types of indexing we consider: *expectational*, *geometric* and *perceptual*. Expectational indexing postulates that reasonable candidates are those which are known to be reachable from the last waypoint visited. Thus we include all waypoints w' for which there exists a link $w_t \overset{a}{\leadsto} w'$. By geometric indexing we mean to find waypoints whose positions are close to the predicted current position of the robot; these waypoints are likely to be good matches. One way of doing this quickly is to index waypoints in a k-d tree [Preparata and Shamos, 1988] by their relative positions (choosing one as an origin). Alternatively, if local reference frames are used for groups of waypoints, a depth-limited search of reference frames near the current waypoint's could be used to find likely waypoint candidates. Finally, perceptual indexing is used to find waypoints with similar views to the current view. We discussed perceptual indexing of image signatures in Chapter 3; similar methods exist for other view representations.

5.2 Mapping Errors

I shall try and correct errors
when shown to be errors;
and I shall adopt new views so fast
as they shall appear to be true views.
—Abraham Lincoln

Viewing the mapping system as an adaptive estimator clarifies its role in the larger picture, as we have seen. However, the incremental nature of learning in this framework means that estimates must be made on-line. In addition, recall that we require that the mapping system operate *passively*, and not actively explore its environment. This is so that mapping will not interfere with pursuit of other robot tasks. However, since the mapper cannot experiment, its decisions must be made with incomplete information. This will lead to erroneous estimates. Since estimation modifies the system's map, erroneous estimation may seriously affect the map. Hence, specific attention must be given to the question of such mapping errors. There are two points that need to be considered: map representations should minimize the problems caused by mapping errors, and methods must be incorporated into the mapper to correct such errors whenever possible. In order to address these points, we must first analyze the types of mapping errors that can occur.

Generally speaking, mapping errors come about due to the fact that the information available at any given time is incomplete and noisy. Information is incomplete because the robot only has a local view of the world; information from a single view is often insufficient for localization. There are a number of varieties of noise that can affect mapping performance. Sensor noise can enter in several ways. Views may be noisy, making them unrecognizable. For example, image signature measurements (see Chapter 3) may vary quite a bit; occasionally this variation will be enough to make two signatures of the same scene unmatchable. Another source of error is the fact that different places may look alike, leading to a disambiguation problem that cannot be handled locally. Also, the same place may look different at different times—if these differences are not modeled (or not detectable), the place may not be recognized. A different sort of sensing error is miscategorization of a waypoint type, such as seeing a T-junction as an L-junction. Included in this category is mistakenly hallucinating a waypoint where none exists[3]. The last type of sensor error relevant to our mapping problem is odometric error. Much of the time, odometric noise can be modeled, if only as bounded error (see Section 2.4.3 for a full discussion). Outliers are still possible, occasionally, when estimated bounds do not hold. All of the above errors can and do occur in ARS MAGNA.

Another source of mapping errors is effector error. There is an implicit assumption that effector modules work as advertised—a wall-following module follows walls, only stopping when the wall ends or its way is blocked. However, developing complete

[3]This is only a problem if the hallucination is inconsistent. If the non-existent waypoint is *consistently* hallucinated, it may be treated for all intents and purposes as a real waypoint by the system.

and correct models of behavior in the real world is essentially impossible. Hence effector modules will occasionally be led astray by the odd cardboard box or wandering graduate students, so actions may sometime have unpredictable and unrepeatable results.

Both sensor and effector errors can cause incorrect information to be entered into a diktiometry directly, for example, by labeling a waypoint node with erroneous perceptual data. However, a more insidious type of mapping error occurs when the system misidentifies the robot's current location. When this occurs, not only is an invalid waypoint description created, but the very structure of the path graph can be affected. In fact, misidentification can occur even without gross sensor/effector errors, simply due to the fact that the system's knowledge is incomplete and measurements are uncertain. This sort of error could be termed 'inferential error', as it is caused by the system making incorrect inferences.

All of these sources of error (sensor, effector, inference) cause a diktiometry to be inconsistent with the real world to some extent. These mapping errors are of several sorts (summarized in Table 5.1). First, there is *positional inconsistency*, caused either by odometric outliers or misidentification of a waypoint. Either a waypoint position estimate or a track position estimate can be in error. Second, there may be *spurious elements* introduced into the diktiometry by sensor or effector errors, or misidentification. Waypoints, views, and action links may all be spurious. And third, we have *structural anomalies*, where the structure of the path graph does not reflect the waypoint structure of the real world. There are two types of structural anomalies: *monotopy*, where multiple nodes represent a single waypoint, and *polytopy* where a single node represents multiple waypoints[4].

As we will see below, all of these error types can eventually be detected, and then corrected. In fact, it is these effects of mapping errors on the structure of the diktiometry that we are really interested in, as they are all we have to go on in diagnosis. As it turns out, with some attention to interactions between error correction techniques, the system does not need to know the exact source of a mapping error in order to correct it.

[4] Think of one (or many) *real* place(s) ($\tau o \pi o \varsigma / \iota$).

Error	Description	Cause(s)
Positional inconsistency:		
Waypoint	Waypoint node's position estimate (relative to some other waypoint in the reference graph) does not contain waypoint's true position.	Misidentification; Odometric outliers
Track	Track's position estimate does not contain robot's true position (relative to track's reference waypoint).	Misidentification; Odometric outliers
Spurious elements:		
Waypoints	Waypoint node imagined where none exists.	Waypoint hallucination; Waypoint miscategorization.
Views	Nonexistent view associated with a waypoint node.	Perceptual outlier; Misidentification.
Action links	Link between two waypoints whose action is unrepeatable.	Effector error.
Structural anomaly:		
Monotopy	Multiple waypoint nodes corresponding to a single real waypoint.	Nonrecognition.
Polytopy	A single waypoint node corresponding to multiple real waypoints.	Misidentification.

Table 5.1: Mapping error taxonomy.

5.3 Prediction

In our diktiometric mapping system, the function of prediction is to provide a rough estimate of the robot's position and orientation, using a previous estimate and new odometric data. All positions and orientations are given relative to some particular reference frame, typically chosen to be that of the track's waypoint. The two components of a track t that are relevant are its position estimate $\mathbf{x}_t$ and its orientation estimate θ_t, both of which are represented as intervals. The relevant sensory data are the odometric position and orientation estimates since the last waypoint, $\mathbf{x}_o$ and θ_o (all intervals, again). The projected position and orientation of the robot are given by the interval equations:

$$\begin{aligned}\mathbf{x}_t^+ &= \mathbf{x}_t + \begin{bmatrix} \cos\theta_t & -\sin\theta_t \\ \sin\theta_t & \cos\theta_t \end{bmatrix} \mathbf{x}_o \\ \theta_t^+ &= \theta_t + \theta_o\end{aligned}$$

If evaluated directly by interval arithmetic, however, this would lead to gross overestimation in $\mathbf{x}_t^+$, because of dependence in the interval rotation matrix (see Appendix A). Tighter interval bounds can be derived by observing that every point in a rectangle is on a straight line segment between two of the rectangle's corners. Hence, to rotate an interval on $\mathrm{I\!R}^2$ by an angle interval Θ, it suffices to rotate the interval's corners (x_i, y_i), giving a set of four 2D intervals $C(\Theta, x_i, y_i)$ (one for each corner). The total rotated interval is just $\sqcup_i C_i$ (the least bounding interval (LBI) of the corner intervals). To rotate a point (x, y) by an angle interval $\Theta = [\underline{\theta}, \overline{\theta}]$, we perform extremal analysis on the equations $x'(\theta, x, y) = x\cos\theta + y\sin\theta$ and $y'(\theta, x, y) = -x\sin\theta + y\cos\theta$. We get the extrema for x at $\theta_x^i = \arctan\frac{y}{x}$, and for y at $\theta_y^i = \arctan -\frac{x}{y}$. We can thus calculate $C(\Theta, x, y)$ as:

$$\begin{aligned}\mathbf{x}'(\Theta, x, y) &= [\min(x'(\underline{\theta}, x, y), x'(\overline{\theta}, x, y), \min_i x'(\theta_x^i, x, y)), \\ &\quad \max(x'(\underline{\theta}, x, y), x'(\overline{\theta}, x, y), \max_i x'(\theta_x^i, x, y)) \quad] \\ \mathbf{y}'(\Theta, x, y) &= [\min(y'(\underline{\theta}, x, y), y'(\overline{\theta}, x, y), \min_i y'(\theta_y^i, x, y), \\ &\quad \max(y'(\underline{\theta}, x, y), y'(\overline{\theta}, x, y), \max_i y'(\theta_y^i, x, y) \quad] \\ C(\Theta, x, y) &= (\mathbf{x}'(\Theta, x, y), \mathbf{y}'(\Theta, x, y))^\top\end{aligned}$$

We now get the predicted track position (the LBI of a set S is denoted $\Box S$):

$$\mathbf{x}_t^+ = \mathbf{x}_t + \Box\left(C(-\theta_t, \underline{x_o}, \underline{y_o}) \cup C(-\theta_t, \underline{x_o}, \overline{y_o}) \cup C(-\theta_t, \overline{x_o}, \underline{y_o}) \cup C(-\theta_t, \overline{x_o}, \overline{y_o})\right)$$

The predicted robot position is the basis of the predicted position of the robot's current waypoint. If waypoints were mathematical points, $\mathbf{x}_t^+$ would constitute an accurate prediction of the current waypoint's position. However, since in the real world waypoints take up a finite amount of space, the waypoint's position is uncertain with respect to the robot's. If we call the maximum radius of a waypoint[5] r_W, we have the waypoint position prediction

$$\mathbf{x}_w^+ = \mathbf{x}_t^+ + \begin{pmatrix} [-r_W, r_W] \\ [-r_W, r_W] \end{pmatrix}$$

[5]This is trivially generalized to allow for different radii for different waypoint types.

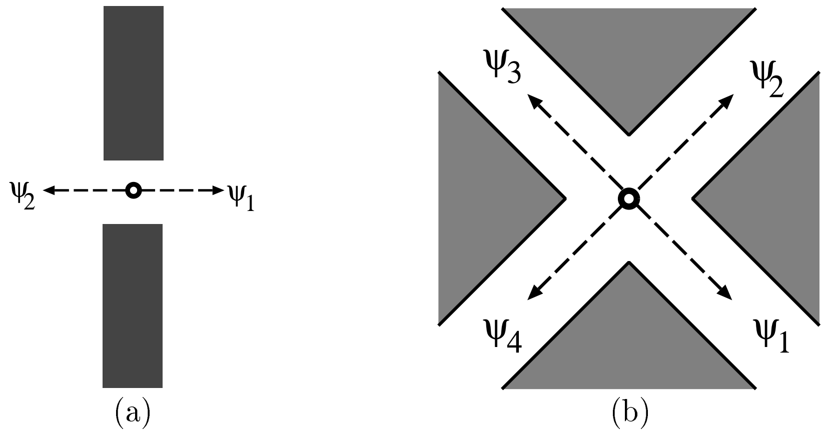


Figure 5.2: Waypoint symmetry. (a) Two symmetries of a doorway. (b) Four symmetries of an X-intersection.

Similarly, the orientation of the waypoint also depends on the orientation of the robot relative to that waypoint. This can be taken into account explicitly, using the robot's sensed orientation relative to the new waypoint, ϕ (an interval with uncertainty). The predicted waypoint orientation is then just $\theta_w^+ = \theta_t^+ - \phi$. The story doesn't end there, however, because of *waypoint symmetry.* Some waypoints are symmetric so that determining relative orientation is inherently ambiguous. The simplest example is the doorway (see Figure 5.2(a)) which has two equivalent local frames of reference which cannot be directly distinguished by sensing. This is not a problem when an unknown waypoint is encountered, since whichever local frame is assumed when sensing can be used as the waypoint's local frame, so disambiguation is unnecessary. However, when a predicted waypoint orientation must be matched to an estimated waypoint orientation, symmetries must be taken into account. This is done by using a model of each waypoint type which represents the waypoint type's symmetries as a set of angles $\Psi = \{\psi_i\}$, eg, $\{0, \pi\}$ for a doorway. The true predicted waypoint orientation is $\Theta_w^+ = \theta_w^+ + \Psi$, represented as a set of intervals. The intersection(s) of this set of intervals with a waypoint node's position estimate can then be computed, as depicted in Figure 5.3. In most cases, this should suffice to disambiguate which waypoint symmetry was assumed by sensing. If, however, orientation uncertainty is great enough, or the waypoint type has many symmetries, all that can be done is narrow down the possibilities. If the intersection of a previous orientation estimate with the

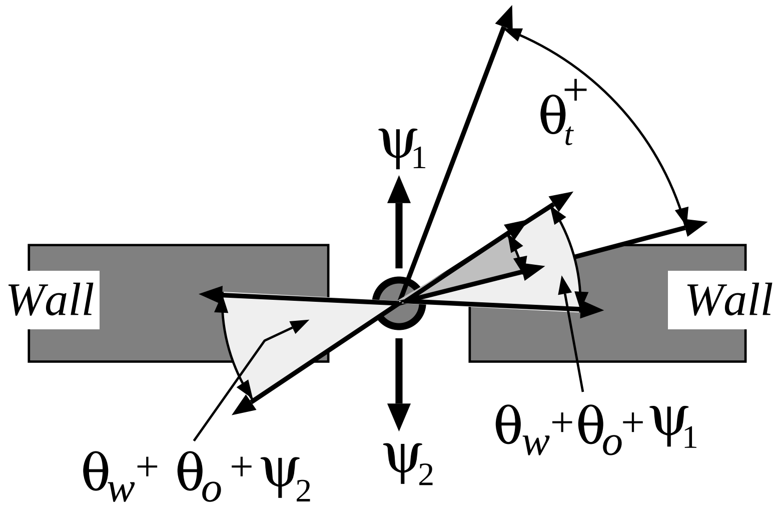


Figure 5.3: Waypoint orientation update for a doorway, showing the intersection of predicted and prior waypoint orientation estimates. For clarity, this figure shown the track update $\theta_t^+ \cap \theta_w - \theta_o - \Psi$ instead of the mathematically simpler waypoint orientation update, $\Theta_w^+ \cap \theta_w$ (θ_w is the prior waypoint orientation estimate).

prediction Θ_w^+ is desired, the least bounding interval can be used for updating. This leads to some overestimation, but it is inevitable if space is to be kept constant[6].

These models of waypoint symmetry can also be made uncertain, to account for natural variation. So, for example, L-intersections may sometimes deviate from right angles. This can be dealt with by using intervals to represent waypoint symmetries ψ_i. These intervals are then treated exactly as above, using interval arithmetic to determine orientation matching and update. Furthermore, if it is clear which symmetry is being used for sensing (i.e., only one ψ_i gives $\theta_w^+ + \psi_i \cap \theta_w$), model uncertainty for the waypoint can be updated by:

$$\psi_i^+ = [\max(\underline{\psi_i}, \underline{\theta_w^+} - \overline{\theta_w}), \min(\overline{\psi_i}, \overline{\theta_w^+} - \underline{\theta_w})]$$

This rules out those elements of ψ_i which are incompatible with the current observation. This cannot be done if more than one symmetry is compatible with an

[6]This scheme could easily be extended to constant-size sets of intervals, say two or three. But this did not prove necessary for our application.

observation, since only the true one can be constrained, and there is no way to tell which it is.

5.4 Matching and Update

The core of the estimator is the process of matching, which produces new estimates of the robot's location based on old estimates, actions performed, and new observations. Matching can also cause the map to be updated, as noted above. The system contains several matching methods, each of which corresponds to a particular way of explaining observations. An arbitration scheme is used to decide which explanations are most reasonable, and should be used. This section describes a general matching engine, as well as the specific matching rules and preference criteria implemented in the mapping system.

5.4.1 The matching engine

Consider the case where the robot encounters a new waypoint, when it has a single current track. Prediction from that track yields a new position and orientation estimate for the robot. Sensory inputs consist of the new waypoint type, a view, the robot's orientation relative to the waypoint, and a label for the action (sequence) the robot just performed. Suppose we consider a single candidate waypoint node. In order to match the robot's current state to that node, an explanation must be found that places the robot at the waypoint represented by that node. This explanation process is carried out implicitly by abduction (inference to the best explanation). There are several different types of situations in which it is reasonable to conclude that the robot is at a particular waypoint (some of which involve previous mapping errors). Each of these situations implies particular relationships between observations and stored information, which the system can test for. If the test succeeds, the explanation may be applicable, in which case a new estimate is constructed based on the explanation, and the map is (possibly) updated. An explanation type, then, corresponds to a *matching rule* or *matcher*, consisting of a *match test*, a *track updater*, and a *waypoint updater*. The test checks to see if the explanation may be applicable, the track updater specifies the new robot state estimate based on the explanation, and the map updater describes how (if at all) the map should be modified based on the match.

More than one kind of explanation may be consistent, so only the most reasonable should be chosen. This is handled by a set of *preference rules*, which specify which matchers are to be preferred to which. The rules are situational, in that the relative plausibility of different explanations varies according to the system's state of knowledge about the area of the world it is in. This means that when multiple tracks exist, the preference relation depends on which track we consider (since different tracks predict different robot locations). How these preferences are determined is discussed below in Section 5.5, where we describe the rules used in the mapping system. If the preference rules introduce cycles, where A is preferred to B is preferred to C is preferred to A, all preferences involved in such cycles are removed, starting with the smallest. The existence of such a cycle indicates that the rules which created it are not sufficient to distinguish between the matchers involved, and so the matchers cannot be distinguished on this basis.

The preference rules place the matchers in a partial order so that only the maximally preferred matchers whose tests are satisfied are allowed. In what follows, m will denote an arbitrary matcher, test_m its test (returning a boolean), $\mathsf{update}_m^{\mathsf{T}}$ its track updater, and $\mathsf{update}_m^{\mathsf{W}}$ its waypoint updater; $<_t$ will denote the (transitively closed) partial order on matchers induced by the preference rules assuming track t correct, where $m_1 <_t m_2$ when m_1 is preferred to m_2. We say that m' is a child of m if m is directly preferred to m'. We will now discuss the several components of the matching subsystem in detail.

5.4.2 Matching rules

There are several matching rules in the system, each corresponding to a different way of explaining observations. The decision to create a new waypoint node is also implemented as a matcher, since it arises from explaining observations by postulating a hitherto unknown waypoint. The rules, their tests, and their updaters are described below, and summarized in Table 5.2.

Continue

Suppose the system's map is complete and correct. Then the only matcher needed for proper estimation should be (modulo rare outliers) one which confirms the robot's location at an expected waypoint. Since actions can be nondeterministic, of course, a choice must still be made on the basis of sensor data. This explanation ("I'm where I expected to be") gives rise to the **Continue** matcher. This matcher tests to see if the current waypoint type, view, and projected odometric position estimate are consistent with the candidate waypoint, and that a link from the last estimated waypoint to the candidate exists, labeled with the last action taken. If these conditions are satisfied, the waypoint's position estimate is updated to its intersection with the predicted position estimate. The track which was **Continue**d is extended to be at that waypoint with its new position estimate.

Link

A matcher that is similar to **Continue** is **Link**. If applied, this matcher decides that the robot is at a known waypoint, but that it got there via an unknown action link. **Link**'s test is identical to **Continue**'s, with the exception that **Link** does not require an action link to exist to the waypoint candidate. Updating is also similar to **Continue**, except that the waypoint updater also adds a new action link, labeled by the last action taken by the robot.

Create

The third matcher that does not involve error correction is **Create**, whose explanation is that the robot is at a previously unknown waypoint. Since this is always a valid explanation (if other possibilities are ignored), its test is vacuously true. Of course, this matcher will only be applied if better possibilities do not exist. Its update creates a new waypoint node and action link, labeled with the current sensory data, odometric estimates, and action label data.

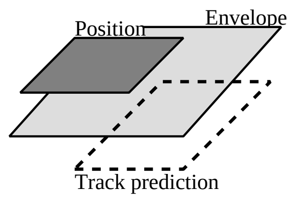


Figure 5.4: **E-Match** test.

Envelope-Match

The first error correcting matcher we will discuss is **E-Match**, designed to cope with waypoint position inconsistency (refer to Table 5.1). To do this, we will make use of a waypoint node's envelope, which, recall, is a position estimate guaranteed to eventually contain the waypoint's true position. Consider the effects of position inconsistency for a waypoint w. Suppose that at some time after the position inconsistency is introduced, the robot arrives at w. It is possible that the robot's predicted position is still consistent with the node's (erroneous) position estimate $\mathbf{x}_w$, due to odometric uncertainty, but let us consider the case where this does not happen. Then, the robot's prediction position interval will not intersect the waypoint node's position estimate interval, but will intersect the waypoint node's envelope (provided it has stabilized), since we assume the track position interval to be correct. So, when this situation occurs, position inconsistency may be reasonably diagnosed.

To summarize, **E-Match** explains an observation by hypothesizing a previous position inconsistency error for a candidate waypoint. It tests first of all that current sensory information (views and waypoint type) are compatible with the candidate. It further tests that the robot's predicted position is inconsistent with the candidate's position, but consistent with the candidate's envelope. The matcher's update attempts to reset the waypoint node's state to a consistent state by expanding its position estimate to include its envelope. The position estimate will get slimmed down by further observations. The track's updated position estimate is the intersection of its prediction with the new, expanded waypoint position estimate.

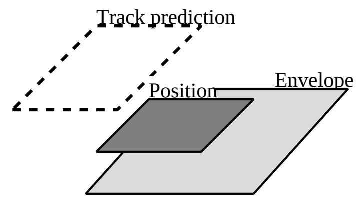


Figure 5.5: **N-Match** test.

Nearby-Match

The second error correcting matcher in the system is **N-Match**, designed to correct for track position inconsistency. The diagnostic situation in this case is that the predicted track position is close to (but not intersecting) the candidate waypoint node's position estimate. This would be expected in a case of track position inconsistency, provided that positional errors are infrequent and not very large, and waypoints are reasonably dense in the environment. These conditions imply that a track whose position estimate is off should only diverge slowly from the robot's true position; hence the track's position estimate should be close to the position estimate of the waypoint that the robot is at. The test for a **N-Match**, therefore, checks (as usual) views and waypoint types, and checks that the minimum distance between the candidate's position interval and the predicted track's position interval is below a threshold. The waypoint node is not updated at all, since the track was in error; the track's position is updated to coincide with the waypoint node's position estimate.

5.4.3 The matching algorithm

On each estimation cycle, prediction produces a set of track predictions T from the last cycle's tracks. Indexing, as described in Section 5.1.3, produces a set $\mathcal{C}$ of candidate waypoints for matching. The general matching algorithm works in two phases. The first (mark) phase computes a set of possible matches, $\mathrm{PM}(m)$, for each matcher, where

$$\mathrm{PM}(m) = \{\langle w, t\rangle \mid w \in \mathcal{C}, t \in T, \mathsf{test}_m(w, t)\}$$

Matcher	Test	Waypoint Update	Track update
Continue	$\mathbf{x}_w^+ \cap \mathbf{x}_w \neq \emptyset$ $\Theta_w^+ \cap \theta_w \neq \emptyset$ $v \in \mathcal{V}_w$ $\tau^+ = \tau_w$ $a^+ \sim a_l$	$\mathbf{x}_w \leftarrow \mathbf{x}_w^+ \cap \mathbf{x}_w$ $\mathbf{e}_w \leftarrow \mathbf{e}_w \cup \mathrm{mid}(\mathbf{x}_w^+)$ $\theta_w \leftarrow \Theta_w^+ \cap \theta_w$	$\mathbf{x}_t \leftarrow \mathbf{x}_t^+ \cap \mathbf{x}_w$ $\theta_t \leftarrow \theta_t^+ \cap \theta_w$
Link	$\mathbf{x}_w^+ \cap \mathbf{x}_w \neq \emptyset$ $\Theta_w^+ \cap \theta_w \neq \emptyset$ $v \in \mathcal{V}_w$ $\tau^+ = \tau_w$ $\nexists a_l \text{s.t.} a^+ \sim a_l$	$\mathbf{x}_w \leftarrow \mathbf{x}_w^+ \cap \mathbf{x}_w$ $\mathbf{e}_w \leftarrow \mathbf{e}_w \cup \mathrm{mid}(\mathbf{x}_w^+)$ $\theta_w \leftarrow \Theta_w^+ \cap \theta_w$ $\mathtt{make} - \mathtt{link}(w_t \overset{a^+}{\rightsquigarrow} w)$	$\mathbf{x}_t \leftarrow \mathbf{x}_t^+ \cap \mathbf{x}_w$ $\theta_t \leftarrow \theta_t^+ \cap \theta_w$
Create	True	New node w $\mathbf{x}_w \leftarrow \mathbf{x}_w^+$ $\mathbf{e}_w \leftarrow \mathrm{mid}(\mathbf{x}_w^+)$ $\theta_w \leftarrow \theta_w^+$ $\mathtt{make} - \mathtt{link}(w_t \overset{a^+}{\rightsquigarrow} w)$	$\mathbf{x}_t \leftarrow \mathbf{x}_t^+$ $\theta_t \leftarrow \theta_t^+$
E-Match	$\mathbf{x}_w^+ \cap \mathbf{x}_w = \emptyset$ $\mathbf{x}_w^+ \cap \mathbf{e}_w \neq \emptyset$ $\Theta_w^+ \cap \theta_w \neq \emptyset$ $v \in \mathcal{V}_w$ $\tau^+ = \tau_w$	$\mathbf{x}_w \leftarrow \mathbf{x}_w \sqcup (\mathbf{x}_w^+ \cap \mathbf{e}_w)$ $\mathbf{e}_w \leftarrow \mathrm{mid}(\mathbf{e}_w) \sqcup \mathrm{mid}(\mathbf{x}_w^+)$	$\mathbf{x}_t \leftarrow \mathbf{x}_t^+ \cap \mathbf{e}_w$ $\theta_t \leftarrow \theta_t^+ \cap \theta_w$
N-Match	$\mathrm{dist}(\mathbf{x}_w^+, \mathbf{x}_w) \leq \vartheta_N$ $\Theta_w^+ \cap \theta_w \neq \emptyset$ $v \in \mathcal{V}_w$ $\tau^+ = \tau_w$	None	$\mathbf{x}_t \leftarrow \mathbf{x}_w$ $\theta_t \leftarrow \theta_t^+ \cap \theta_w$

Table 5.2: Match rule summary. The predicted state of the robot consists of its predicted position $\mathbf{x}_t^+$ and orientation θ_t^+, the new view v, waypoint type τ^+, and the label of the last action executed a^+. The waypoint candidate being considered is w. For **Continue**, the label of the action link that predicted the candidate is a_l.

Each possible match is a pair $\langle w, t\rangle \in \mathcal{C} \times T$ of a waypoint and track. In the second (sweep) phase, the most preferred matches are selected, and their updaters computed. Given PM(m) for all matchers (easily computed), the matching algorithm is as follows:

`Match`(T, $\mathcal{C}$)

1. WU $\leftarrow\emptyset$; TU $\leftarrow\emptyset$;
2. For each track prediction $t^+ \in T$:
   (a) Unmark all matchers
   (b) $Q \leftarrow \{m \mid \nexists m' <_{t^+} m\}$, as a FIFO queue (*order doesn't matter*);
   (c) Loop while Q not empty:
      i. $m \leftarrow$ pop(Q);
      ii. If m is unmarked, For all $\langle w, t^+\rangle \in$PM($m$):
         A. Mark all matchers m' s.t. $m <_{t^+} m'$;
         B. WU $\leftarrow$ WU $\cup\{\mathsf{update}^{\mathsf{W}}_m(w, t^+)\}$;
         C. TU $\leftarrow$ TU $\cup\{\mathsf{update}^{\mathsf{T}}_m(w, t^+)\}$.
      iii. Else $Q \leftarrow$ insert(children(m),Q)
3. Return WU, TU.

This algorithm produces updaters for all potential matches for each track that are not shadowed by a more preferred potential match (for any track). Consider, for example, a case where there are two tracks, t_1 and t_2, such that t_1 has a **Link** match, and t_2 has only a **Create** match. Provided that **Link**$<_{t_2}$**Create**, only the **Link** on t_1 will be allowed. However, suppose that (for whatever reason) t_2 does not prefer **Link** to **Create** (though t_1 might). Then, the **Create** for t_2 is not shadowed by t_1's **Link**, and it should take place. The implications of track ambiguity are slightly more involved than this example indicates, and are discussed below.

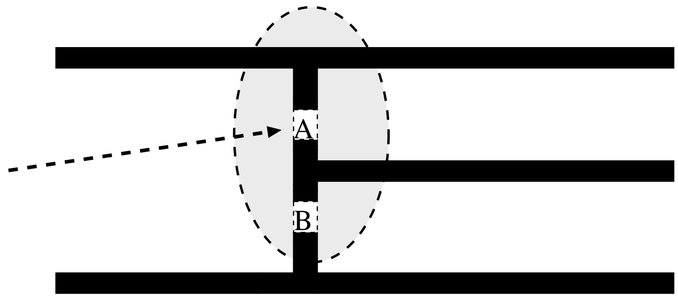


Figure 5.6: An inherently ambiguous world, given that the world looks the same from waypoints A and B, and odometric error is too high for geometry to reliably distinguish them (as shown by the dashed ellipse).

5.4.4 Ambiguity and update caching

Due to sensing and odometric errors, it can often be the case that the mapping system cannot tell which waypoint the robot is at. This can occur even with a perfect map, as in the situation depicted in Figure 5.6. In this situation, the robot will often be unable to tell waypoints A and B apart, no matter what it does (without experimentation), since the waypoints look alike and odometric error is high. However, after moving on, the robot may be able to tell where it was, if, for example, it performs a sequence of actions with different known results for the two possible starting waypoints. As long as the ambiguity persists, however, changes to the map should not be performed, since they may be based on incorrect assumptions about the robot's location. Therefore, waypoint updates should be cached whenever multiple hypotheses about the robot's position exist, i.e. when matching produces multiple tracks. When the robot's location is eventually disambiguated, the updates can then be performed.

There are three sorts of transformations that track estimation can cause, in addition to the straightforward case of one track producing a single new estimate: *divergence*, *convergence*, and *evaporation* (depicted in Figure 5.7).

- Divergence occurs when one track has more than one preferred estimate—this is where ambiguity is introduced. When a track diverges, each estimate's waypoint update is cached on an update queue associated with the estimate. If a new waypoint is created for the track, it is marked as cached and placed on a

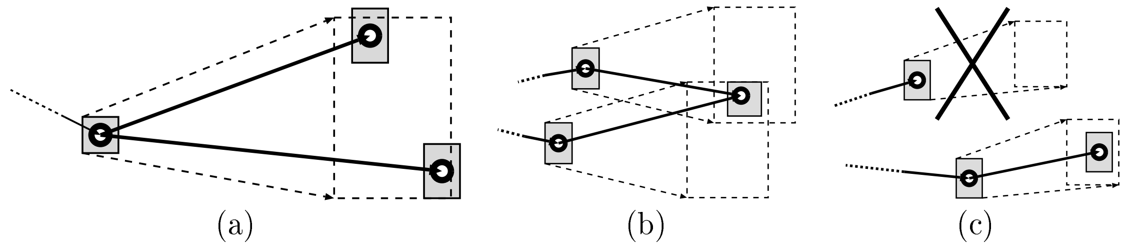


Figure 5.7: Track transformations. Tracks are shown as circles, track position predictions by dashed lines, and waypoint position estimates by grey rectangles. (a) Divergence. (b) Convergence. (c) Evaporation.

waypoint cache list for the track. Cached waypoints may only be matched to tracks for which they are cached. Both new tracks inherit cached updates and waypoint from the original track.

- Convergence is when two different tracks are both extended to the same waypoint. Ambiguity about the robot's current location is removed, but convergence makes it impossible to tell anything about the past. Therefore both tracks' update caches are cleared. If only one track remains, all cached waypoints are confirmed. If multiple tracks remain, than the converged track inherits all the cached waypoints from the original tracks.

- Finally, evaporation occurs when a track has no preferred estimates. The track is simply removed. When all tracks but one have evaporated or converged, the waypoint updates in the remaining track's update cache are applied to the map. Only the most recent cached update for each waypoint is applied, since different updates for a waypoint may be inconsistent, as later updates are calculated without earlier updates having been applied. Also, all created waypoints which are cached on an evaporated track and not also cached on a non-evaporated track are destroyed.

If several tracks exist, and if they are each extended uniquely, the waypoint updates (or waypoint creations) that result are cached on the tracks, as for divergence.

5.4.5 Disjunctive updates

As mentioned above, more than one waypoint updater may apply to a single waypoint node. This may occur because two tracks converge on the node, or because two incommensurable matchers match the node. We treat all such circumstances uniformly, and merge the waypoint updaters generated into a single *disjunctive* updater. Each updater specifies new constraints on the state of the waypoint under consideration, so we simply disjoin all of the new constraints to get the most general interpretation of the new situation. This means that the hull union of all the position (or orientation) updates is used. Similarly, all new links suggested are added to the map, since incorrect links will eventually be elided (see Section 5.6.1 below), and some incorrect links will not seriously affect performance. The basic idea is, as elsewhere, to use the principle of least commitment.

5.5 Match preferences

In order to arbitrate between different matchers which may all be applicable in a given situation, the system contains *preference rules* which describe when certain matchers are to be preferred to others. In theory, these preference rules could use many different sorts of information to determine which instantiated matchers are to be preferred to which others. Complicated probabilistic models could be used, for example, which would calculate the likelihoods of the different matches, given observations (and perhaps the robot's history). Designing such models would, however, be very difficult to do in any principled way. Hence, we restrict the information used by preference rules to two sorts: matcher type and match context. Note that the track and candidate waypoint node that are matched are not used. This is both for simplicity, and because specific matches of different types are not easily commensurable. In any case, the simpler approach worked in our system. By 'match context' we mean information that the system gathers over time relating to its state of knowledge. For example, one sort of 'contextual knowledge' is how well-visited an area of the world is. If it is reasonable to assume that the robot is somewhere it has been before (even if it is not sure exactly where) matchers which accord with that assumption will be preferred to matchers which contradict it. The rest of this section will describe the preference rules used by our system (for the matchers described above). Contextual

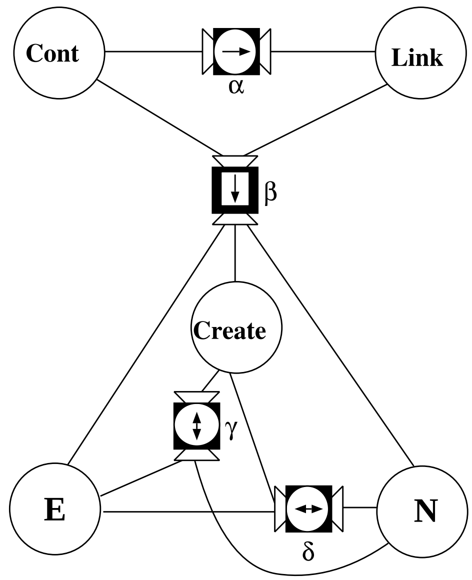


Figure 5.8: Matcher preference rules, as a graph. Rule tests are as follows: α if well-traversed; β always; γ if and only if well-visited; and δ if and only if track confident. See text for a full explanation.

knowledge is implemented in the system by track confidences and confidence grids, described further below.

5.5.1 Preference rules

Most of the preference rules used in the system work by computing a value and comparing it to a threshold. Some of the rules are 'reversible', in that if the value is below a low threshold, the preference goes in one direction, and if it is above a high threshold, the preference goes the other direction. The middle range is taken to represent the state of 'not enough evidence'. There are four heuristic preference rules in the system (relating sets of matchers to sets of matchers). The preference rules are depicted graphically in Figure 5.8. They will be stated here in intuitive terms, the precise methods used are described below Sections 5.5.2 and 5.5.3.

First, **Continue** and **Link** are always preferred to other matchers, since we assume that the map is correct in the absence of evidence to the contrary. Second, when the area just traversed by the robot has been extensively traversed by the robot before, it is unlikely that a new link will be discovered, so **Continue** is preferred to **Link**. (The implementation of deciding whether an area is 'extensively traversed' is taken up further below.) If the area has not been extensively traversed, there is no reason to prefer **Continue** over **Link**, or vice versa. The last two preference rules control how the map is modified. One rule prefers an **E-Match** or a **N-Match** to a **Create**, if the robot believes that it has probably visited its current location before. If it did, it did not just discover a new waypoint. Conversely, if it is very unlikely that it has visited its current location before, **Create** is preferred to either **E-Match** or **N-Match**. Finally, **E-Match** and **Create** are preferred to **N-Match** if the track under consideration has a high confidence; the reverse preference holds if the track has a particularly low confidence. Track confidence is discussed next; determining 'visitedness' is discussed in the following section.

5.5.2 Track confidence

In order to decide whether a **N-Match** is appropriate, the system needs some notion of how likely it is that a track is incorrect ('off-track', so to speak). Unfortunately, the two main factors which cause this, erroneous position update and odometric outliers, are not directly observable, so we have to use a heuristic approximation. This is not as problematic as it may seem, as exact track confidence is not important; all we need to be able to do is roughly compare track correctness. The method is based on the types of the matchers used to update a track. Each track is associated with a confidence measure c. When the track is extended, the confidence(s) of its estimate(s) c^+ are calculated as functions of the track's previous confidence, c^-, and the matcher type used. We divide our matchers into three types: *confirming* (**Continue** and **Link**), *agnostic* (**Create**), and *disconfirming* (**E-Match** and **N-Match**). Confirming matchers increase our confidence in a track, since they demonstrate the consistency of the track with the map (assumed to be correct unless proven otherwise). **Create**, on the other hand, does not affect our confidence in a track's correctness, since it introduces no information not already present in the track. Finally, matchers which are used to diagnose past mapping errors serve to disconfirm tracks, for two reasons. First, abductive error diagnosis is not entirely certain, and such matches may be due to track errors as well as mapping errors. Secondly, **N-Match** directly diagnoses the existence of a track error. Even though **N-Match** is meant to correct the track error it diagnoses, its effectiveness may only be judged in hindsight, by future confirmations.

These ideas are implemented by (loosely) interpreting track confidence as the probability that the track is correct, i.e, that its position estimate contains the robot's true position. On a **Create** match, $c^+ = c^-$ since no new information is introduced. Each other matcher m is associated with a confidence parameter p_m, which is interpreted differently for confirming and disconfirming matchers. For confirming matchers, p_m is the probability that the extended track will be correct after a match of type m, regardless of other factors[7]. Conversely, a disconfirming matcher's confidence parameter p_m indicates the probability that a correct track remains correct after m is applied[8]. Thus, the update formulae for our four matchers are:

$$
\begin{array}{lrcl}
\mathbf{Continue:} & c^+ & = & 1-(1-c^-)(1-p_C) \\
\mathbf{Link:} & c^+ & = & 1-(1-c^-)(1-p_L) \\
\mathbf{E-Match:} & c^+ & = & c^- p_E \\
\mathbf{N-Match:} & c^+ & = & c^- p_N
\end{array}
$$

Based on the way individual matchers operate, we can indicate ordering constraints on the values of their confidence parameters:

- $p_C > p_L$, since **Continue** indicates greater consistency with the map than **Link**;
- $p_E > p_N$, since **E-Match** affects track position more conservatively than **N-Match**, and since **N-Match** indicates a previous track position error.

To make this order total, we specify that $p_L > p_E$, which is reasonable since we expect most non-creation matches to be **Continue**s or **Link**s. We set the values a priori at 0.95, 0.9, 0.8, and 0.75, respectively, and no further tuning was required. A track confidence c was considered high if $c > 0.85$ and low if $c < 0.3$. There is no reason to believe that tuning these parameters is critical to system performance.

---

[7]We thus use a zero-order Markov model of track confirmation. Higher-order models could be devised, conditioning the probability of correctness by previous match types, but it is not at all clear how to assign the conditional probabilities.

[8]This should be seen more properly as an update of our *belief* that it is correct; applying the matcher may change our confidence by changing our belief about what happened in the past.

5.5.3 Confidence grids

The second heuristic for determining match preferences is to estimate how likely it is that the robot has been somewhere before. Since matching is only done at waypoints, a sparse set of positions in the environment, we distinguish two tests: if the robot has registered a waypoint at its current position before (*visitation*), and if the robot has moved from its last position to its current position before (*traversal*). They must both be represented, since visitation of two waypoints does not mean that the robot has ever moved between them. We represent this information approximately, using a variation of certainty grids [Elfes, 1987]. Space is tessellated by a *confidence grid*[9], with each cell (x, y) associated with a *visitation confidence* $v(x, y)$ and a *traversal confidence* $t(x, y)$. These numbers are meant to represent the confidence the robot has that it has either visited, or passed through, a point in the given grid cell before. Since we have eschewed in this work the use of full statistical models of robot motion and odometry due to the practical difficulty of obtaining such (as discussed in Section 2.4.3), we do not attempt to interpret confidences here as probabilities, but rather as relative confidence measures.

Every time a track is extended, the estimate's interval position estimate, $\mathbf{x}_t$, is used to update the confidence grid for the current reference frame. As for waypoint updates, if multiple tracks coexist, updates are cached until the ambiguity is resolved. However, confidence grid updates are not discarded on track convergence, rather the different updates are all applied, weighted by the inverse of the number of converging tracks (assuming that they were all equally likely).

For visitation, we assume, in the absence of any other statistical assumptions, a uniform distribution of robot position over $\mathbf{x}_t$. (Perhaps a truncated normal distribution would work better, but the uniform approach has proved sufficient thus far.) If A is the area of $\mathbf{x}_t$, $v(x, y)$, $(x, y) \in \mathbf{x}_t$, is updated by:

$$v(x, y) \leftarrow 1 - (1 - v(x, y))(1 - 1/A)$$

This says, in effect, that a cell has been visited if it was visited before, or if it was just visited now. Given this confidence grid, we can now determine our confidence that the robot has previously visited its current position, estimated by position interval $\mathbf{x}_e$ with area A_e. We sum across all positions (x, y) in $\mathbf{x}_e$ the probability of the

[9] Actually, one confidence grid is associated with each local reference frame.

robot currently being at (x, y) and having previously visited (x, y). We again make the simplifying assumption that the robot's current position is distributed uniformly across $\mathbf{x}_e$, giving us the visitation confidence:

$$V_e = \frac{1}{A_e} \sum_{(x,y)\in\mathbf{x}_e} v(x, y)$$

A similar method is used to determine traversal confidence. Ideally, since we want to decide whether the robot has likely gone between two particular points before, we should use a 4-dimensional certainty grid, where each cell (u, v, x, y) contains the confidence that the robot has gone between (u, v) and (x, y). However, this method is too space intensive; a resolution of 40 grid cells in each dimension would require over 2.5 megabytes of storage (assuming each confidence value is represented by a single byte). Therefore we make an approximation and use a 2-dimensional grid, where we calculate confidences that the robot ever traversed each grid cell before (in any direction). Since we are not interested in the exact path the robot took between two points, just if it ever moved between them before, we can pretend that its path was a straight line. We thus use the envelope of all straight line paths between the previous track position estimate, $\mathbf{x}_t^-$, and $\mathbf{x}_t$, which is an irregular convex hexagon (call it H), in general. Let its area be B, and its maximum diameter (the longest possible traversal distance) d. The confidence of any particular cell in H having just been traversed is then d/B. We update $t(x, y)$, $(x, y) \in H$ just as we did $v(x, y)$ above:

$$t(x, y) \leftarrow 1 - (1 - t(x, y))(1 - d/B)$$

Retrieval is somewhat more complicated, however. Let $\mathbf{x}_e$ be the current observed track position estimate, and $\mathbf{x}_e^-$ the previous track position estimate. Ideally, we should examine all possible paths between points in $\mathbf{x}_e^-$ and $\mathbf{x}_e$ separately; however this would be quite costly, with complexity $\mathrm{O}(dn^4)$ (where $n = \lceil \frac{1}{r} \max(\mathrm{rad}(\mathbf{x}_t), \mathrm{rad}(\mathbf{x}_t^-))\rceil$). We simplify matters first by using a *sliding area* method (illustrated in Figure 5.9). We calculate the chance of an uncertainty interval $\mathbf{x}$ (of area A) representing a traversed cell, as above:

$$T(\mathbf{x}) = \frac{1}{A} \sum_{(x,y)\in\mathbf{x}} t(x, y)$$

We now consider intervals $\mathbf{x}_\alpha$ $(0 \leq \alpha \leq 1)$, interpolated between $\mathbf{x}_0 = \mathbf{x}_e^-$ and $\mathbf{x}_1 = \mathbf{x}_e$. The minimum of $T(\mathbf{x}_\alpha)$ over α, T_e, is taken as our confidence of traversal

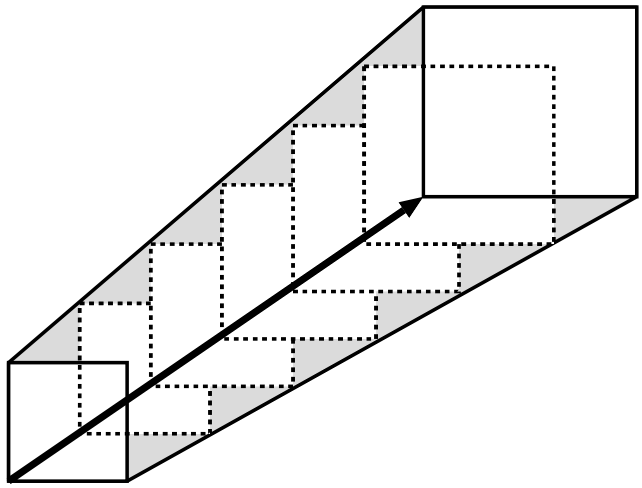


Figure 5.9: The sliding area method for traversal retrieval. See text for explanation.

between $\mathbf{x}_e^-$ and $\mathbf{x}_e$. This measure does not have a direct probabilistic interpretation (even approximately, as for visitation), since cell traversal could be in any direction, even transverse to that desired. It can be justified, however. Suppose T_e is large. This implies that $T(\mathbf{x}_\alpha)$ is large along the entire path from $\mathbf{x}_e^-$ to $\mathbf{x}_e$. Assuming that traversals are sparse (likely in structured environments), we would not expect $T(\mathbf{x}_\alpha)$ to be continuously large due to transverse traversals, since paths cross each other sparsely (assuming robot trajectories are mostly nearly straight). Thus we can conclude that the likelihood of a past traversal is high. Conversely, if T_e is small the likelihood of a past traversal is small, since it cannot be larger than T_e (since a past traversal implies a traversal through $\mathbf{x}_\alpha$ for all α). The sliding area method has complexity $\mathrm{O}(dn^2)$.

Example

Examples of coverage grids produced during map-learning (under control of a pseudo-random walk) are shown in Figure 5.10(b) and (c). The robot spent most of its time in rooms C and D. The visitation coverage grid shows the robot's concentration on the middle door and its neighboring corners, as well as the end of the protruding wall. The doors are also represented as well visited. In the traversal grid in Figure 5.10(c)

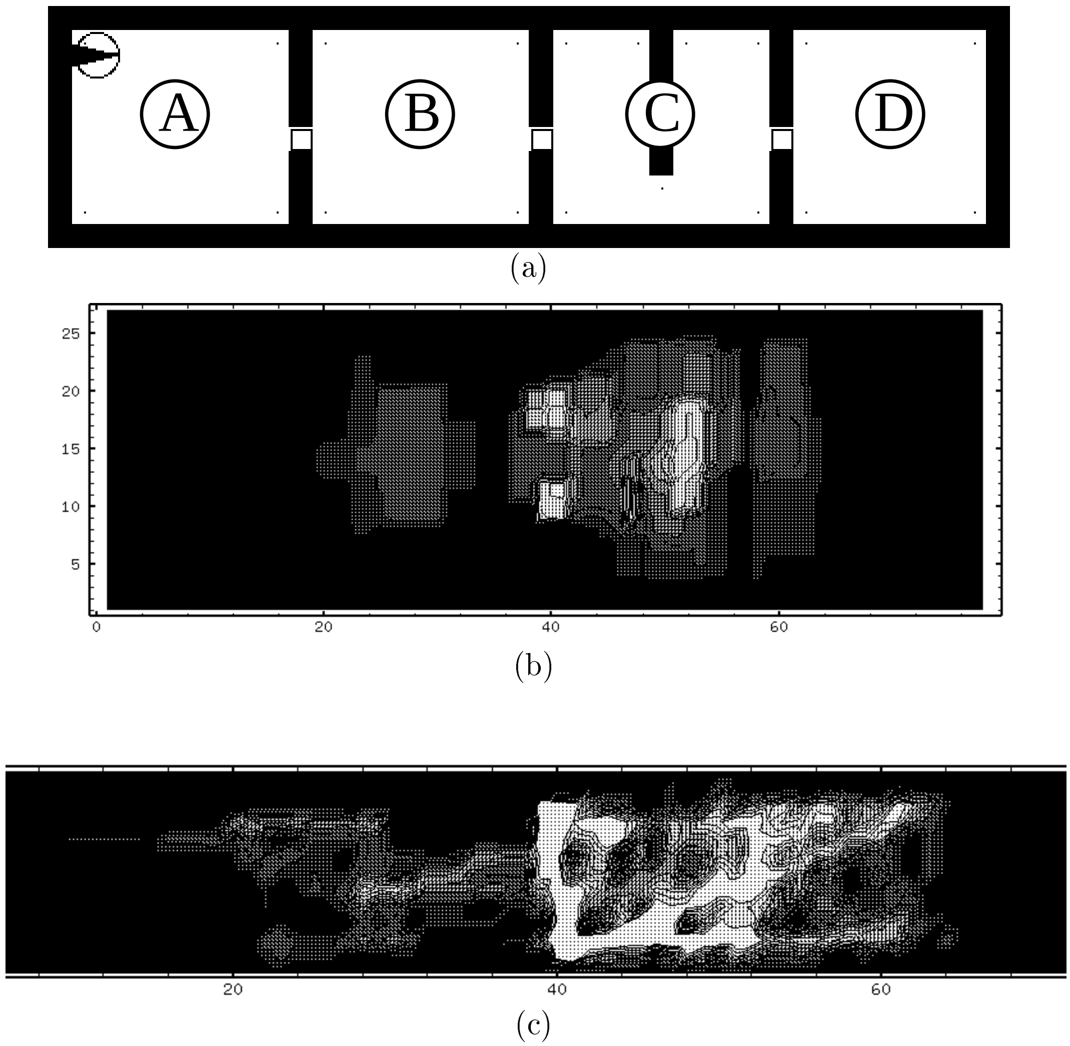


Figure 5.10: Examples of traversal and visitation coverage grids after 200 waypoint visits under a random walk. Scales on the plots give x and y grid cells. (a) The test environment. (b) Visitation coverage grid. Lighter colors represent more visitation. (c) Traversal coverage grid. Lighter colors represent more confidence in prior traversal.

we see more clearly the structure of the robot's behavior in room C (and, to a lesser extent, room D). The robot often moved between the leftmost corners in the room and the left door, but also moved between the lower-left corner and the protruding wall, and from there to the right door. On the other hand, the robot less often moved from the protrusion to the lower-right corner (probably because the wandering algorithm is biased towards doorways). This example shows how coverage grids are able to capture the long-term structure of the robot's behavior.

5.6 Restructuring

Nine times out of ten,
in the arts as in life,
there is actually no truth to be discovered;
there is only error to be exposed.
—H. L. Mencken

In the last sections, we discussed error diagnosis and correction by means of explanatory matching rules. These kinds of rules can be used to diagnose errors which result in instantaneous inconsistency, where single observations can indicate inconsistency with previous mapping decisions. Positional inconsistency is of this type, as we saw. However, spurious elements and structural inconsistency cannot be diagnosed instantaneously. Rather, we need to accumulate information over time, and reinterpret old observations (i.e, diagnose mapping errors) when the information so indicates. We term this sort of long-term map update *restructuring*. The remainder of this section describes restructuring strategies for diagnosing and correcting mapping errors due to introduction of spurious elements or structural anomalies into a map.

5.6.1 Spurious elements

The simpler problem is that of spurious elements introduced into the map. Spurious elements are introduced into the map by observations whose invalidity cannot be determined at observation time. Hence we must integrate information over time to determine which observations were valid and which were not. The essential idea we use here is the fact that spurious elements, by their very nature, are infrequently observed.

The method we use then is one of *transient elision*, in which we maintain counts of how often each feature f (a view, link, or waypoint node) is observed ($\#O(f)$) and how often it is expected ($\#E(f)$). When a feature is encountered far less than it is expected, it may be classified as transient, and deleted. A view v on a waypoint w is expected whenever w is visited. If we assume that views should be seen with approximately equal frequency, we want to elide v if $\#O(v)\,|\mathcal{V}_w| < \epsilon_v \#E(v)$, where $\mathcal{V}_w$ is w's view set and ϵ_v is a threshold parameter < 1. An action link $l = w_1 \overset{\alpha}{\rightsquigarrow} w_2$ is expected whenever α is executed from w_1, and observed whenever a **Continue** match is performed on l. Just as for views, l is elided if $\#O(l)|\{w_1 \overset{\alpha}{\rightsquigarrow} w_i\}| < \epsilon_l \#E(l)$. A waypoint node w is expected either if a link into w is expected or if the robot's predicted position estimate is consistent with w's. In addition to comparing the frequency of observation of a waypoint node, we also wait until most action links to the waypoint have been elided.

5.6.2 Structural anomaly

Recall the two types of structural anomaly we consider: *monotopy*, where one waypoint is represented by multiple nodes in the map, and *polytopy*, where two (or more) waypoints are represented by a single node. We only deal here with the primitive case of two waypoints (or nodes); when more waypoints (or nodes) are involved, the additional elements will be taken care of individually as evidence accumulates. When monotopy is diagnosed, the two waypoint nodes involved should be merged; conversely, when polytopy is diagnosed for a waypoint node, it should be split into two.

Merging

We will present below several diagnosis techniques; in all cases merging and splitting proceeds similarly. We will discuss merging first. Assume we have decided to merge waypoint nodes w_1 and w_2 to form composite waypoint node w. First, w's action links are the union of those of w_1 and w_2; equivalent links (by label and destination) are merged. Similarly, w is assigned the union of w_1 and w_2's view sets. Finally, we merge their geometric constraints. If the waypoints have multiple reference links, we only consider those to points that they share. We can now consider the case of combining constraints on w_1 and w_2 with respect to a single point p; the procedure is followed for all such constraint pairs independently. Let the positions of w_1 and w_2 with respect to p be intervals $\mathbf{x}_1$ and $\mathbf{x}_2$ respectively. Then, if $\mathbf{x}_1 \cap \mathbf{x}_2 \neq \emptyset$, the constraints are consistent and should be combined, giving a position constraint on w, $\mathbf{x} = \mathbf{x}_1 \cap \mathbf{x}_2$. If, on the other hand, $\mathbf{x}_1$ and $\mathbf{x}_2$ are not mutually consistent, we must be more conservative, and take their hull union, $\mathbf{x} = \mathbf{x}_1 \sqcup \mathbf{x}_2$. Dealing with orientation constraints, of course, is somewhat more complicated due to waypoint symmetry. We have no way of knowing with respect to which of its symmetries we have a waypoint's orientation estimate, hence w_1 and w_2 may describe the same waypoint, but may be oriented to different symmetries and so have widely differing orientation estimates θ_1 and θ_2. The answer is to consider all pairs of orientation estimates θ_1^i, θ_2^i, where $\theta_j^i = \theta_j - \psi_i$ and $\{\psi_i\} = \Psi(\tau_w)$ are the waypoint symmetries for the waypoints' type. Note that each θ_j^i is a legitimate orientation estimate for w_j. We choose the closest such pair θ_1^* and θ_2^*, where distance between angle intervals is defined by:

$$\mathrm{dist}(\theta_1, \theta_2) = \begin{cases} -|\theta_1 \cap \theta_2| & \text{if } \theta_1 \cap \theta_2 \neq \emptyset \\ \min_{a,b \in \theta_1, \theta_2} |a - b| & \text{otherwise} \end{cases}$$

The new orientation estimate for w is then $\theta^* = \theta_1^* \cap \theta_2^*$, if it is non-empty, since we combine consistent estimates; otherwise, we have conflicting information, so we take the hull union as above, $\theta^* = \theta_1 \sqcup \theta_2$.

Splitting

Splitting is slightly more complicated than merging. Call the waypoint node to be split w. The splitting algorithm takes two inputs from diagnosis, one required, the other optional. The required argument is a pair of *modes* p_1 and p_2 to split on, each a point position, both with respect to the same reference frame. The other argument consists of two disjoint (possibly empty) subsets of action links on w, A_1 and A_2, meaning that the action links in A_i should be assigned to resultant waypoint w_i (splitting always produces two waypoints). The non-geometric aspect of splitting is simple: all action links not in $A = A_1 \cup A_2$ are copied, with one copy going to each of the new waypoints ('children'); then A_1 and A_2 are assigned to their respective children. Views are treated similarly, those associated exclusively with links in A go to their relevant waypoints, all others are copied to both children. Those views or

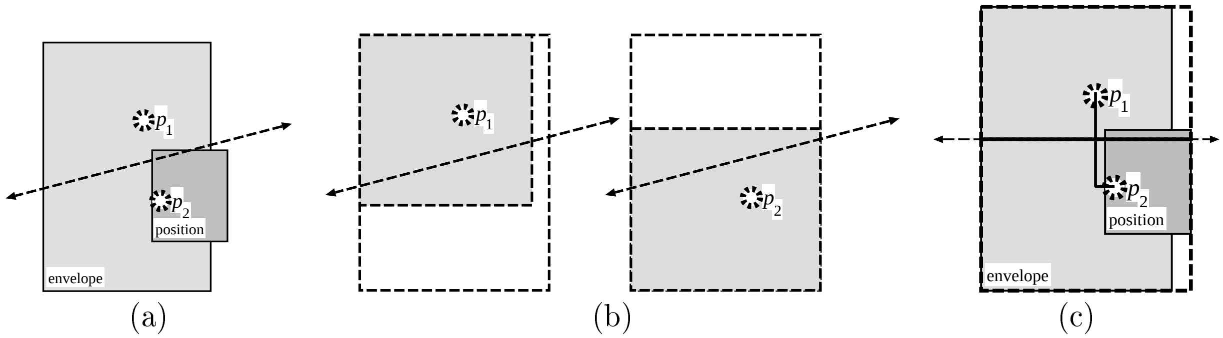


Figure 5.11: Waypoint splitting. (a) Splitting modes and the equidistant line. (b) Resultant interval position estimates for the children; note the overlap. (c) Axis parallel splitting heuristic.

action links improperly assigned to one of the children are treated in future processing as transient hallucinations, and will be elided. Both children get the same waypoint orientation estimate from w.

The core of splittage is the splitting of waypoint position estimates based on p_1 and p_2. We start working within a single reference frame, and then describe how we make all other constraints consistent with our splittage (if multiple reference frames are used, as described in Chapter 4). We wish to derive new position estimates (intervals) for w's children. We base these estimates on p_1 and p_2, meant to be the best point estimates for w_1 and w_2 determined by diagnostic critics. We first look at the hull union of w's position estimate and envelope, X, since the envelope is asymptotically guaranteed to contain the true positions of all waypoints used to update w (i.e, w_1 and w_2). We then wish to split the interval X along the line equidistant between p_1 and p_2 (see Figure 5.11(a)). This leads to some overestimation when we move back to intervals, as seen in the figure; it also makes the estimates for the children overlap. This overlap was seen to be a problem in practice, since it allowed more frequent misidentifications. Therefore, the system was modified to split X parallel to an axis. We split along the axis-parallel line bisecting the longer axis-parallel distance between p_1 and p_2 (see Figure 5.11(c)). This gives us the most accurate disjoint approximation to the original splitting method. The envelopes of the children are reinitialized to contain only p_1 and p_2, respectively, since w's envelope includes points from both the (assumed) waypoints w_1 and w_2. We call the position estimates derived for the new waypoints $\mathbf{x}_1$ and $\mathbf{x}_2$.

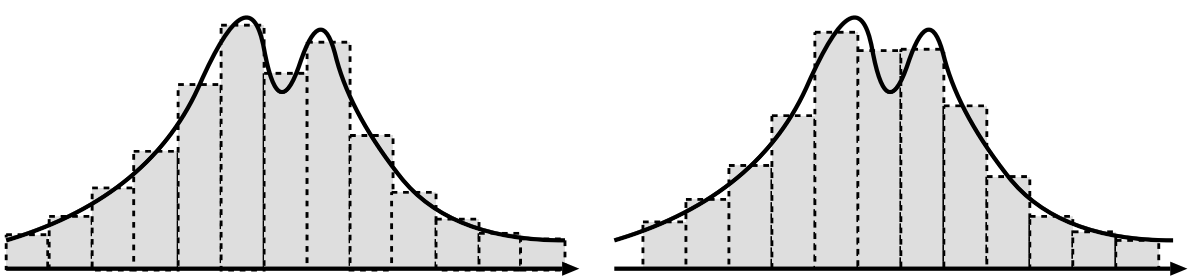
Let the reference frame assumed here be frame F_i. There is now no guarantee that the new ${}^i\mathbf{x}_1$ and ${}^i\mathbf{x}_2$ are consistent with prior position constraints in other reference frames (since we used w's envelope in F_i). We adjust those other constraints for w_1 as follows (the procedure for w_2 is the same). Let ${}^j\chi$ be a constraint on w's position with respect to F_j, $j \neq i$, and let ${}^j\mathbf{x}_1 = {}^jF_i{}^i\mathbf{x}_1$ be the image of $\mathbf{x}_1$ in F_j. Suppose that ${}^j\mathbf{x}_1 \cap {}^j\chi \neq \emptyset$, i.e, the prior constraint is consistent with the split-derived constraint. Then we simply update the constraint to include only the consistent portion thereof, ${}^j\chi' = {}^j\mathbf{x}_1 \cap {}^j\chi$. Otherwise, the prior constraint is inconsistent with the new constraint, which takes priority, so the old constraint is discarded.

Positional diagnosis

The first diagnosis method we discuss for mono/polytopy is based on the statistics of waypoint position estimates. The main assumption that we use is *waypoint separability*: we postulate that waypoints cannot be arbitrarily close to one another; we term the minimum waypoint distance δ_{sep}. (We could just as easily have a different minimum distance for different waypoint types; the algorithms described here could be used almost as is.) Note that, in any case, two waypoints certainly cannot be closer to each other than either the diameter of the robot and the minimum waypoint size. We add to this two statistical assumptions. First, that the probability distributions for waypoint position estimates are unimodal. The second assumption is used for the purposes of diagnosing polytopy. We assume that waypoints that may be confused with one another (they are close and look similar) are visited with near-equal frequency, and so that their combined position estimate distribution remains multimodal. If this is not the case, then it is impossible to geometrically separate distinct waypoints without stronger distributional assumptions. This assumption is a conservative one, since if it is violated, we only get false negatives (no split) and can fall back on other diagnosis methods. Using stronger invalid statistical assumptions (such as Gaussianity) for diagnosis could result in incorrect splittage. These assumptions allow us to use bimodality testing on position estimates to decide whether to split (bimodal) or merge (unimodal).

Recursive stochastic hillclimbing

There are a number of different methods in the statistical literature for testing the bimodality of the probability distributions of a random variable (here waypoint position estimates). We restrict our attention to non-parametric methods, which do not assume particular types of distributions (as in, for example, work on mixture models [McLachlan and Basford, 1988]). One simple type of method for checking if a probability distribution is bimodal is examining a histogram of observations—multiple maxima in the histogram provide evidence of bimodality (provided enough samples are collected). However, as Figure 5.12 shows, use of histograms is sensitive to the quantization used to produce them. Another non-parametric bimodality test uses the minimum spanning tree of the observations to decide between unimodality and bimodality [Ramey, 1982]. The idea is that the links of the tree are shorter in the vicinity of a mode (since more observations will fall there); tests can be devised to determine if two 'short-link regions' exist separated by longer links or not. In this section, we will present a new non-parametric method for testing bimodality (and finding modes), called *recursive stochastic hillclimbing* (RSH).



(a) (b)

Figure 5.12: Quantization effects in histograms. Bimodality is clear in (a), but completely obscured by shifting the origin in (b).

Suppose we want to find the mode of a unimodally distributed random variable X (with density f_X). We can view this as a function maximization problem: Find x maximizing $f_X(x)$. The simplest method (absent analytic characterization of f_X) is gradient hillclimbing: Starting with an initial estimate $e(0)$, move it as $e(t+1) = e(t) - \alpha \nabla f_X(e(t))$ until convergence. Unfortunately, we cannot directly evaluate

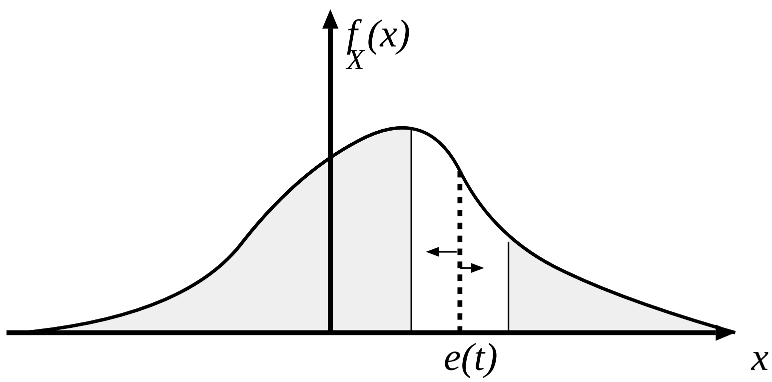


Figure 5.13: Simple stochastic hillclimbing. $e(t)$ moves in response to observations in the white region nearby.

∇f_X, or even f_X; all we get are samples of X (call them $z(t)$). Consider, though, the following stochastic estimation algorithm:

$$e(t+1) = \begin{cases} (1-\alpha)e(t) + \alpha z(t) & \text{if } \|z(t) - e(t)\| \leq \delta \\ e(t) & \text{otherwise} \end{cases}$$

The operation of this algorithm in the one-dimensional case is depicted graphically in Figure 5.13. If the observation is near the current estimate, the estimate is moved in the direction of the observation. We can easily see that e will tend to move in the direction of increasing f, since there is a greater probability that an observation will fall in that part of the window. We call this movement *stochastic hillclimbing.* We formalize this notion below, and extend the treatment to multimodal distributions.

Let a *mask function* of width ω $m_\omega(x)$ be a positive, symmetric, unimodal function with mode (and mean) at 0, such that:

- $\int_{-\infty}^{\infty} m_\omega(x) dx = 1$;
- $\lim_{w \to 0} m_\omega(x) = \delta(x)$, converging smoothly to a Dirac delta function;
- $\forall \omega > 0, \exists l, h,\ l < 0 < h$, s.t. $l < x < h \Rightarrow m_\omega(x) > 0$.

One example of a mask function in 1 dimension is the uniform pillbox function:

$$u_w(x) = \begin{cases} \frac{1}{2w} & |x| \leq w \\ 0 & \text{otherwise} \end{cases}$$

Another is a Gaussian distribution, whose standard deviation is its width. Given a mask function m, we define the basic recursive stochastic hillclimbing algorithm RSH-1 by:

$$e(t+1) = e(t) + m_\omega(z(t) - e(t))(z(t) - e(t))$$

When the mask function is narrow (ω is small), e will move relatively little, compared to when ω is large. On the other hand, as ω goes to 0, the expected change of e is proportional to the gradient of f_X at e (since m_ω goes to a delta function). This is the use of the mask function—we start with large ω, getting a coarse approximation to the true gradient (in the average movement of e), but quick movement; then we slowly move ω towards 0, as we get closer and closer to the true mode. This method has some inherent 'jitter', so temporal smoothing can be applied by averaging observations over a temporal window.

This method can also be used to find the modes of multimodal distributions (and hence, determine multimodality), by assuming a minimum intermode distance δ_{sep}. To deal with multiple modes, we allow multiple mode estimates, called here *estims* (for lack of a better term). Each estim e_i has a position $x(e_i)$ (its estimate of the mode) and a weight $W(e_i)$ (the number of observations making it up). Each observation $z(t)$ is assigned to the closest estim, which is then updated by weighted averaging according to the rule:

$$\begin{aligned} x(e_i(t+1)) &= \tfrac{W(e_i(t))x(e_i(t)) + m_\omega(z(t) - x(e_i(t)))(z(t) - x(e_i(t)))}{W(e_i(t)) + m_\omega(z(t) - x(e_i(t)))} \\ W(e_i(t+1)) &= W(e_i(t)) + 1 \end{aligned}$$

The algorithm is started out with one estim at $z(0)$. At each time step, if $z(t)$ is far from all estims ($\|z(t) - x(e_i(t))\| > \delta_{sep}$, for all i), a new estim e' is created with $x(e') = z(t)$ and $W(e') = 1$. Conversely, when two estims e_i and e_j are closer than $\delta_{sep}/2$, the estims are merged, giving a new estim e' which is the weighted average of e_i and e_j:

$$\begin{aligned} x(e') &= \tfrac{W(e_i)x(e_i) + W(e_j)x(e_j)}{W(e_i) + W(e_j)} \\ W(e') &= W(e_i) + W(e_j) \end{aligned}$$

We want to maintain the invariant that all estims are more than $\delta_{sep}/2$ distance apart, so that they represent (potentially) different modes. Doing this can require combining more than two estims at a time. The following algorithm to merge all estims in an estim set E, by repeating it until E is returned unchanged:

1. For all $e_i \in E$: let $S(e_i) = \{e_i\}$;
2. For all $e_i, e_j \in E$:
   - For all $e_j \neq e_i \in E$:

     If $\|x(e_i) - x(e_j)\| \leq \delta_{sep}/2$:
     - $E \leftarrow E \cup \{e'\}$, where
       $x(e') = \frac{W(e_i)x(e_i)+W(e_j)x(e_j)}{W(e_i)+W(e_j)}$,
       $S(e') = S(e_i) \cup S(e_j)$, and
       $W(e') = \sum_{e \in S(e')} W(e)$;
     - Remove e_i and e_j from E;
     - Go to the next e_i in Step 2.
3. Return the new estim set E.

This algorithm iteratively merges all mergeable estim pairs, keeping track of the correct weights by support sets (the sets $S(e)$). The full recursive stochastic hillclimbing algorithm RSH (with a fixed mask width[10]) is:

1. Initialize $E = \emptyset$.
2. Get a new observation z;
3. If there exists estim $e \in E$ s.t. $\|z - x(e)\| \leq \delta_{sep}$, then:
   $x(e) \leftarrow \frac{W(e)x(e)+m_\omega(z-x(e))(z-x(e))}{W(e)+m_\omega(z(t)-x(e_i(t)))}$
   $W(e) \leftarrow W(e) + 1$;
4. Else, add a new estim e' to E, where:
   $x(e') \leftarrow z$
   $W(e') \leftarrow 1$;

[10]A fixed mask width is sufficient for our purposes, as we only need accuracy to within $\delta_{sep}/2$.

5. Merge all mergeable estims in E;

6. Go to Step 2.

As observations come in, all estims are expected to move towards the modes of the distribution. We thus expect estims near the modes to eventually accumulate considerably more weight than estims elsewhere. Our test for bimodality, then, is to determine if two estims, e_1 and e_2 exist, such that (for $i \in \{1, 2\}$; $\rho, \vartheta_e > 1$):

$$\begin{array}{lrcl} \forall e' \in E(w), e' \neq e_i & W(e_i) & > & \rho_w W(e') \\ & W(e_i) & > & \vartheta_e \\ & \max_i W(e_i) & \leq & \rho_w \min_i W(e_i) \end{array}$$

We see if (a) these two estims dominate all others, (b) that they have a minimum amount of supporting evidence (at least ϑ_e observations each), and (c) that neither dominates the other. This indicates that the two estims correspond to two modes of the observation probability distribution. If, on the other hand, only one dominating estim is found, the distribution is probably unimodal.

We justify the RSH algorithm based on the results of a simulation on uni- and bimodally distributed random variables in two dimensions. The distributions used were mixtures of Gaussians with a standard deviation of 2. The unimodal variable was distributed by a single Gaussian of mean $(0, 0)$, while the bimodal case was an even mixture of Gaussians with means $(0, 0)$ and $(5, 0)$. We ran the RSH algorithm with a pillbox mask of fixed width 5 for 10,000 observations and got the following results:

Test case	Three weightiest estims (x;w)		
Unimodal	(0.13,0.05);13990	(-3.86,3.13);264	(14.64,-9.07);2
Bimodal	(-0.07,0.22);8263	(5.50,-0.17);7285	(5.51,6.29);25

As the table shows plainly, the difference between the unimodal and bimodal cases is easily discerned, as are the approximate locations of the modes. (The weights do not add up to 10,000, since an observation may be merged with two different estims, with the results not mergeable.) The use of a fixed-width pillbox mask introduces some uncertainty into the mode estimates, but for our purposes this uncertainty is not unreasonable.

We now discuss how positional cues are used for diagnosing mono- and polytopy.

Monotopy

The geometric assumptions discussed previously are used to diagnose monotopy by looking for pairs of waypoint nodes, w_1 and w_2, of the same type which are likely to be at the same position. We present two methods. A simple method diagnoses monotopy when the hull of the waypoint nodes' position estimates has maximum diameter less than δ_{sep}, $2\,\mathrm{rad}(\mathbf{x}_1 \sqcup \mathbf{x}_2) < \delta_{sep}$. This means that the two nodes' waypoints must be closer than δ_{sep}, hence, by waypoint separability, the nodes correspond to the same waypoint. This situation is depicted graphically in Figure 5.14(a).

The simpler method will rarely be applicable (since it requires low estimate uncertainty), so we also use RSH for merge diagnosis. We diagnose monotopy if $\mathbf{x}_1 \cap \mathbf{x}_2 \neq \emptyset$, and w_1 and w_2 individually appear to have unimodal position estimates with estimated modes within $\delta_{sep}/2$ of each other. Translated into terms of estims, we determine if there exist estims $e_1 \in E(w_1)$ and $e_2 \in E(w_2)$ such that (for $i \in \{1, 2\}$):

$$\begin{array}{lrcl} \forall e' \in E(w_i), e' \neq e_i & W(e_i) & > & \rho_w W(e') \\ & W(e_i) & > & \theta_w \\ & \|x(e_1) - x(e_2)\| & < & \frac{1}{2}\delta_{sep} \end{array}$$

If these conditions are satisfied, then monotopy is diagnosed, and we merge w_1 and w_2 as described above. This situation is depicted graphically in Figure 5.14(b).

Polytopy

In order to diagnose polytopy, we can use RSH to determine if a waypoint's position distribution has multiple modes at least δ_{sep} distant from each other. We thus have the condition that a waypoint node w has two strong estims e_1 and e_2 with similar weights, with at least δ_{sep} distance between them (for $i \in \{1, 2\}$):

$$\begin{array}{lrcl} \forall e' \in E(w), e' \neq e_i & W(e_i) & > & \rho_w W(e') \\ & W(e_i) & > & \theta_w \\ & \max_i W(e_i) & \leq & \rho_w \min_i W(e_i) \\ & \|x(e_1) - x(e_2)\| & > & \frac{1}{2}\delta_{sep} \end{array}$$

When these conditions are satisfied, w is split, with mode estimates $p_i = x(e_i)$ and empty action link sets $A_i = \emptyset$, since positional diagnosis gives no information about

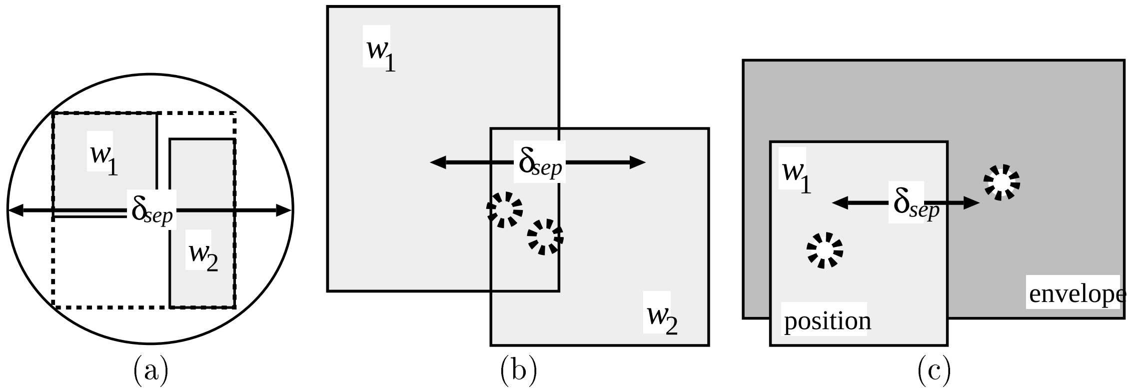


Figure 5.14: Positional diagnosis of structural anomaly. (a) Definitive diagnosis of monotopy. (b) Estim-based diagnosis of monotopy (stars are dominating estims). (c) Estim-based diagnosis of polytopy.

action links. We thus must rely on transient elision to adjust link sets accordingly. The estim set of each child waypoint node w_i is initialized to contain those of w's estims contained within its position estimate $\mathbf{x}_i$ after splitting. Positional polytopy diagnosis is depicted graphically in Figure 5.14(c).

Connective diagnosis

The second method we have developed for diagnosing structural anomaly is based on analyzing the links between waypoints for inconsistencies. The method is based on the intuition that different waypoints connect to the rest of the world in different ways. That is, different waypoint nodes should have different action link sets, and links between each pair of waypoint nodes should be consistent with one another. We say that two links, l_1 and l_2, are *consistent* ($l_1 \sim l_2$) if $\Delta\mathbf{x}_{l_1} \cap \Delta\mathbf{x}_{l_2} \neq \emptyset$. We ignore link labels here, since all ways of going between two points results in the same change in position. Now, if two nodes correspond to a single waypoint (monotopy), we would expect them to (eventually) have all the same links to the rest of the world. Similarly, if a single node corresponds to two (or more) waypoints (polytopy), we might expect to be able to separate its action links into two (or more) inconsistent subsets. This type of diagnosis is not complete, as very regular environments can confuse it. However, in conjunction with positional diagnosis it works well. The details of the method are described below.

As is generally the case, merging is the simpler diagnosis. Suppose we have two waypoint nodes w_1 and w_2, with respective outgoing link sets $\{l_{1i} = w_1 \rightsquigarrow d_{1i}\}$ and $\{l_{2j} = w_2 \rightsquigarrow d_{2j}\}$. We say that a link l_{1i} from w_1 is consistent with w_2 if there exists a link l_{2j} such that $l_{1i} \sim l_{2j}$ and $d_{1i} = d_{2j}$. Given this, if more than a fraction (in our tests, 80%) of w_1's links are consistent with w_2, and vice versa, we may declare the nodes mergeable. In order to ensure that sufficient evidence exists, we require in addition that both w_1 and w_2 have a minimum number of outgoing links (in our tests, 3).

In order to diagnose polytopy on a node w by link analysis, we need first to find links which are inconsistent with one another. If the links from w to some destination node d can be partitioned into two (or more) sets, such that the intersection of $\Delta\mathbf{x}$ estimates for one set is disjoint from that of the other, we say that d is inconsistent with respect to w. Determining if this is the case is easily done. If enough of the destinations reachable from w are inconsistent, and w has enough action links to support the diagnosis, polytopy may be diagnosed.

Given this diagnosis, we still need to compute the proper modes to split on. In addition, connectivity diagnosis can give us an idea of which action links belong to which children. Each action link $l_i = w \rightsquigarrow d_i$ from w implies a derived position estimate for w, $\mathbf{x}_w^i = \mathbf{x}_d - \Delta\mathbf{x}_{l_i}$. If polytopy has been diagnosed, then for a majority of destinations d, there exist at least two disjoint derived position estimates for w. Let those position estimates for destination d_i be denoted $\mathbf{x}_j^i$. We now find two consistent sets of such position estimates X_1 and X_2, choosing one $\mathbf{x}_j^i$ for as many d_i as possible; call the intersections of the sets $\mathbf{x}_1$ and $\mathbf{x}_2$ respectively. If $\|\,\mathrm{mid}(\mathbf{x}_1) - \mathrm{mid}(\mathbf{x}_2)\| > \delta_{sep}$, splitting is performed on the modes $\mathrm{mid}(\mathbf{x}_1)$ and $\mathrm{mid}(\mathbf{x}_2)$. The action link sets A_1 and A_2 are taken as those links which gave rise to X_1 and X_2. All other links are copied to both children.

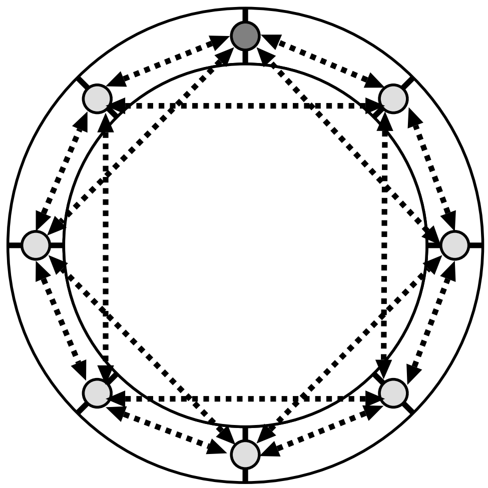


Figure 5.15: Example illustrating the need for both structural diagnosis methods for splittage (see text). Filled circles represent waypoints and dashed arrows links between them.

Discussion

In this section, we have discussed two different methods for diagnosing structural anomaly: positional and connective diagnosis. We use both methods in tandem to correct structural mapping errors, because they are both useful. The need for both of these techniques can be illustrated by the example depicted in Figure 5.15. In this environment, all waypoints except for one are indistinguishable; assume a high level of odometric uncertainty and that waypoints may be missed as the robot moves around. In this case, neighboring waypoints around the ring will be confused with one another, causing polytopy. Since all waypoints are connected to their neighbors identically, connective diagnosis will not suffice to correct the map, so positional diagnosis must be used (with positions implicitly referred to the identifiable waypoint).

Chapter 6

Mapping Results

"Now," said Rabbit,
"this is a Search, and I've Organised it—"
"Done what to it?" said Pooh.
"Organised it. Which means—well, it's
what you do to a Search, when you don't
all look in the same place at once."
—A. A. Milne

6.1 Evaluation methodology

The passive mapping system developed here was evaluated using the Ars Magna simulator described in Chapter 2. (For convenience, we refer to the basic unit of length as meters.) The experiments reported in this chapter tested the performance of the mapping system as the robot performed a kind of random walk through its environment. The random walk procedure was defined in terms of primitive behaviors (such as wall-following or door-traversal), and so was not easily reduceable to a simple random walk on the environment's path graph. The idea of using a random walk is that it approximates a worst-case analysis with respect to different tasks. Most tasks usually result in more-or-less repetitive behavior, which will tend to revisit the same places often and trace similar paths, not to mention use the map as it is constructed. A random walk largely avoids these phenomena.

As maps are learned, we can look at the shape of the diktiometries that are produced, and compare them to the 'real world' to evaluate their accuracy. However, the mere fact that a learned map 'looks right' is no guarantee that it is a good one, since there is quite a bit of hidden information. A map that looks good now may easily become garbage in the future. There are, however, more objective methods for measuring mapper success. The first is based on the concept of expectation error—the error inherent in relying on the map to predict the robot's position after each move. We can measure this by calculating the sum-of-squared-distance (SSD) between the robots actual relative motion and estimated relative motion for each track after a move. We average the SSD over all tracks. If the system is effective at mapping, we expect the average SSD per move (SSD/M) to asymptotically converge on a small constant.

Another measurement of mapping error is the average number of error corrections per move. This can be examined by looking at the average number of map adjustments per move (MA/M). Map adjustments are defined as waypoint creations, **E-Match**, **N-Match**, and waypoint splits and merges. Again, we expect MA/M to converge on a small constant if mapping is effective.

6.2 Mapper behavior

Before we discuss mapping results for specific environments, there are some general qualitative observations that we can make regarding the behavior of the mapping system during learning. Approximate matching and restructuring often worked together to correct errors in surprising ways. In this section we will briefly describe the more prominent of these observed phenomena.

Perhaps the simplest is a coupling of **E-Match** and node splitting. When two nearby waypoints are confused, a single waypoint node is created for both. Its position estimate will tend to converge to a point midway between the two real waypoints. This leads a situation in which a visit to one of the waypoints produces a predicted track position that is inconsistent with the node's position. Usually this results in an **E-Match** being applied, expanding the node's position estimate to include both real waypoints. This sequence (convergence–**E-Match**) may repeat several times until enough data is gathered to allow the node to be split. Sometimes, however,

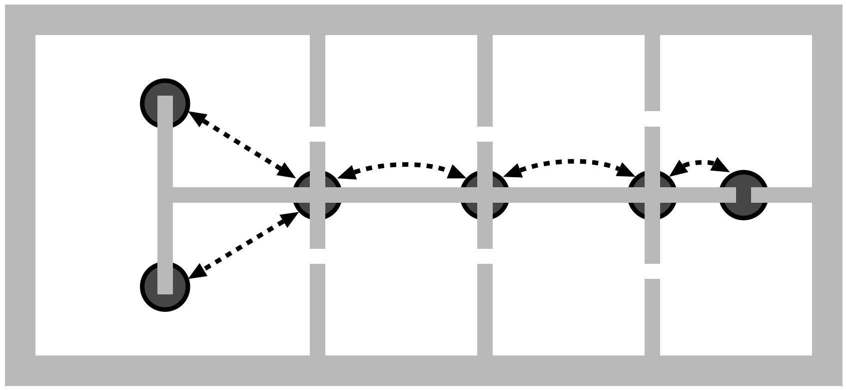


Figure 6.1: The situation in which the zippering phenomenon occurs. The environment is superimposed on the diktiometry path graph; nodes correspond to doorways. Either of the two polytopic nodes on the ends will be split first, followed by the other two.

E-Match is not applicable (because the envelope is not large enough yet), so a new node is created. The first node then either converges to the other waypoint, or is eventually split, with one of the split's children eventually merging with the created node.

Another phenomenon associated with splitting and merging may be termed 'zippering'. This can occur where either (a) several adjacent waypoint nodes are polytopic or (b) several pairs of pairwise adjacent nodes are monotopic (see Figure 6.1). When this is the case, the first nodes to be split or merged are those adjacent to the rest of the diktiometry. Once one split or merge occurs, this enables more accurate position data to be gathered for the other erroneous nodes, and they then proceed to be split or merged. In linear sequences of nodes (as in GreyWorld, see below), this looks like a (very slow) zipper opening or closing.

Finally, we have the phenomenon of 'map crawling'. If two similar nodes are near enough to each other that they may be confused, the first visit to one may produce a spurious identification with the other. This may then lead to a waypoint node whose position estimate is not consistent with any real waypoint. This may then lead to the erroneous creation of waypoint nodes that are not position consistent with any real waypoints. If left unchecked, this would lead to the creation of an entire new map

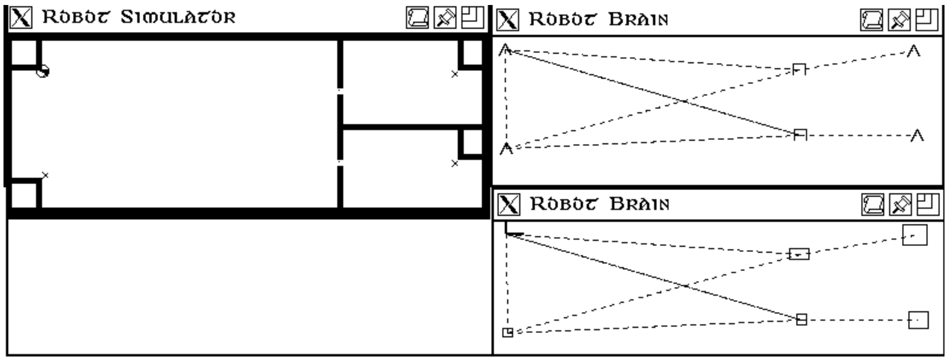


Figure 6.2: GreyWorld and a typical diktiometry learned. All walls are a uniform grey. The upper map picture shows the topology with symbols indicating different waypoint types. The lower picture shows waypoint position estimates.

at the new position, and this crawling can be repeated leading to an infinite cycle of partially known maps (see the lesion study below for an example of this). Crawling is stopped by either **E-Match** or **N-Match**, when the robot comes close enough to a known waypoint node to reregister its location correctly. There does not seem to be any general guarantee that this will suffice, however, so further study of map crawling is warranted.

6.3 Basic results

In this section we will describe the results of two experiments which demonstrate the effectiveness of the passive mapper. The mapper was first tested on two simple environments meant to examine the resilience of the system to environmental ambiguity and variety. In the next section we will discuss the results of mapping in a more complex world, which exercises all of the mapper's faculties together.

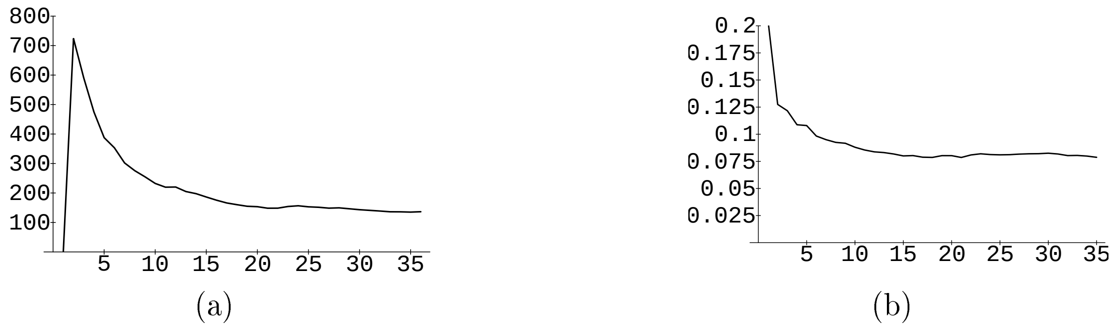


Figure 6.3: GreyWorld error measures, averaged over 10 runs. The x-axis shows number of waypoint encounters, in units of 10 (i.e, from 0 to 350). (a) Cumulative SSD/M (square 'meters'). (b) Cumulative MAO/M (adjustments per match).

GreyWorld

The first world we examine is that depicted in Figure 6.2, named 'GreyWorld'. The right side of the world is symmetric, introducing unavoidable ambiguity into the robot's location estimates. This ambiguity is enhanced by the fact that all the walls are indistinguishable (perhaps painted flat grey, hence the name). This world tests the mapping system's ability to tease apart the two identical doorways and corners at the right. These experiments did not use waypoint orientation, so the leftmost corners were only distinguishable by the robot moving from one to the other. Except where otherwise noted, these experiments were done with 10% odometric position noise and no angular noise.

Quantitative results for mapping in GreyWorld are graphed in Figure 6.3. The graph for SSD/M shows an initial sharp increase, as the robot discovers new waypoints, since position estimates for new waypoints are very uncertain. SSD/M peaks when all waypoints have been encountered, and then it starts to slope down gradually, as position estimates are refined. Also, in this region (between 40 and 110 encounters) the mapper discovers and corrects two instances of polytopy (on the doors and rightmost corners). After this, SSD/M settles down to a relatively smooth convergence to about 135m^2. Note that there is inherent ambiguity in the world, as the robot cannot directly distinguish the two doors and rightmost corners. We can calculate a rough estimate of what the SSD/M would be for a perfect map, assuming an equal

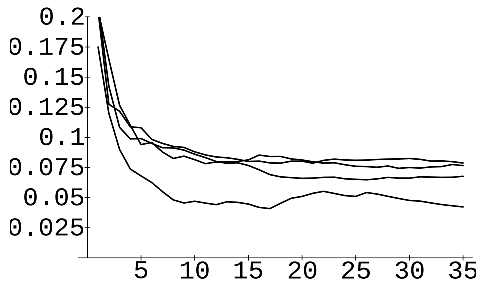


Figure 6.4: GreyWorld cumulative MAO/M with different levels of translational odometric noise (10%, 15%, 20%, and 25% of distance traveled). Data from a single run is plotted for each noise level.

number of visits to each waypoint at equilibrium (this only a rough estimate). All visits to doors or rightmost corners are ambiguous, as are about half the visits to leftmost corners, hence about 5/6 of visits are ambiguous. Each ambiguous encounter gives an SSD/M of $12^2 = 144$ (since the distance between doors is 12m); we thus get an expected final SSD/M of 96. The actual number is a bit higher for two reasons. First, the waypoint position estimates did not converge completely to points (as seen in Figure 6.2); this uncertainty contributed to measured SSD/M. Secondly, the assumption of equiprobable links is not quite accurate—the robot often went back and forth between doors and their associated corners, so the ambiguous waypoints were visited more often than our simple model would indicate.

MAO/M exhibits similar behavior to SSD/M. The number of map adjustments starts out high, as new waypoints are discovered and created. MAO/M drops rapidly in that phase. After about 30 moves, all waypoints have been visited, but (a) their position estimates are uncertain, and (b) the map contains errors (chiefly polytopy of the doors and rightmost corners). The decline in MAO/M is then slower, as map adjustments are used to slowly correct for errors in the map. After about 120–130 visits, MAO/M settles down to a value of about 0.08 adjustments per visit, which is reasonable given the continual introduction of hallucinated action links and occasional false waypoints into the map.

Figure 6.4 shows results for varying levels of odometric noise. The MAO/M curves are qualitatively similar for different noise levels, all indicating satisfactory convergence (the 10% noise run converged atypically well). As noise increases, convergence slows, but not unreasonably so, and map errors remain small.

SpaceWorld

Our second test world is depicted in Figure 6.5. This world has few waypoints spread out over large distances, and its walls are highly distinguishable. In fact, the walls are so variable in appearance that the same waypoint often appears quite different on different visits. This environment, then, primarily tests the ability of the mapping system to deal with large odometric uncertainty and view variability. Error measure plots for SpaceWorld are shown in Figure 6.6. The plots have the same general shape as those for GreyWorld, with some small differences. First, initial map error is far greater, both because of the greater distances (hence odometric noise) in SpaceWorld, and because more waypoints are initially created (since waypoints are not easily recognized). Second, convergence is slower, with the first phase (completeness) lasting until about visit 75, and the second phase (error correction) lasting until visit 250 or so. This is due both to the fact that there are more waypoints in SpaceWorld than in GreyWorld, and the greater number of errors that need to be corrected (some waypoints in SpaceWorld gave rise to four or more waypoint nodes initially). Final errors, though, converge on reasonable values. SSD/M steadily moves towards zero, as waypoint position uncertainty is slowly whittled away. And MAO/M approaches the same asymptote as for GreyWorld above.

6.4 Lesion study

In addition to the experiments described above, which directly tested the ability of the passive mapping system to learn properly, we also performed a small lesion study to examine the effects of not using error correction during map learning. The environment used in this study was the simple one (similar to GreyWorld above) shown in Figure 6.7(a). Translational odometric noise was 10% of distance traveled and rotational noise 5% of (total) amount turned. Two mapping runs were performed: one using the full mapping system, and one excluding **E-Match**, **N-Match**, elision, merging, and splitting (the *naive* mapping system). The naive mapping system is

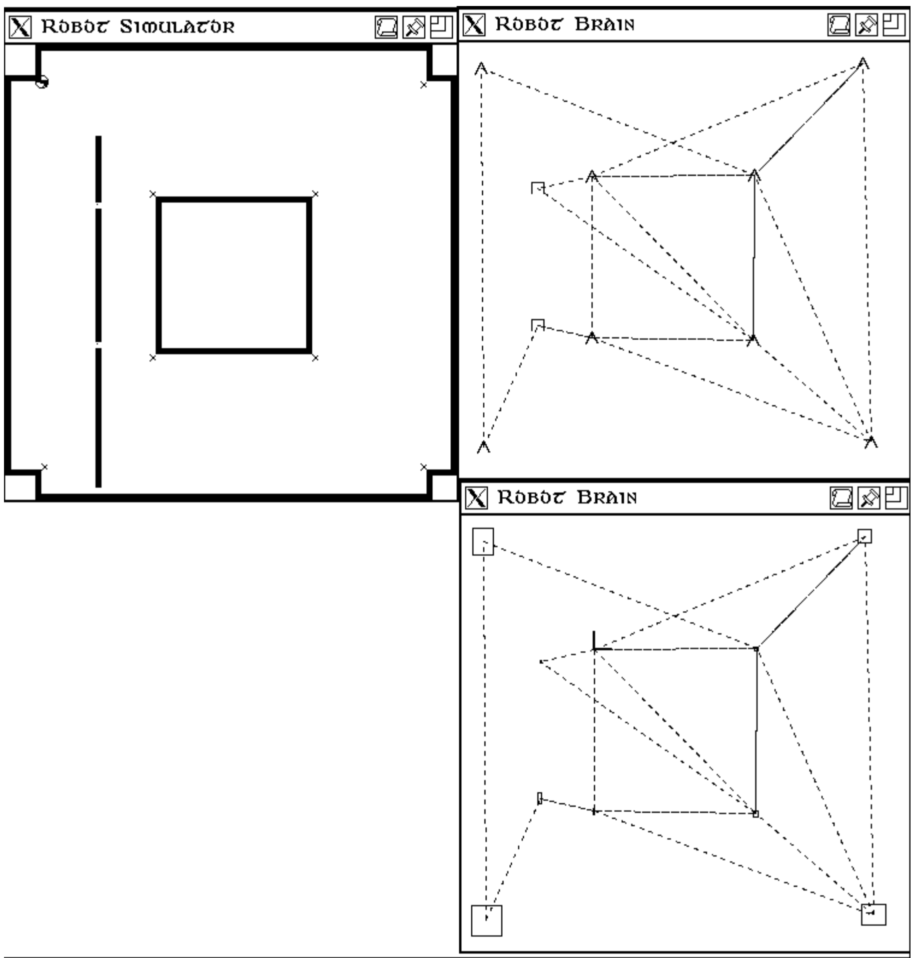


Figure 6.5: SpaceWorld and a typical diktiometry learned. The upper map picture shows the topology with symbols indicating different waypoint types. The lower picture shows waypoint position estimates. Note that a straight line between waypoint nodes implies some known path between them, not necessarily straight.

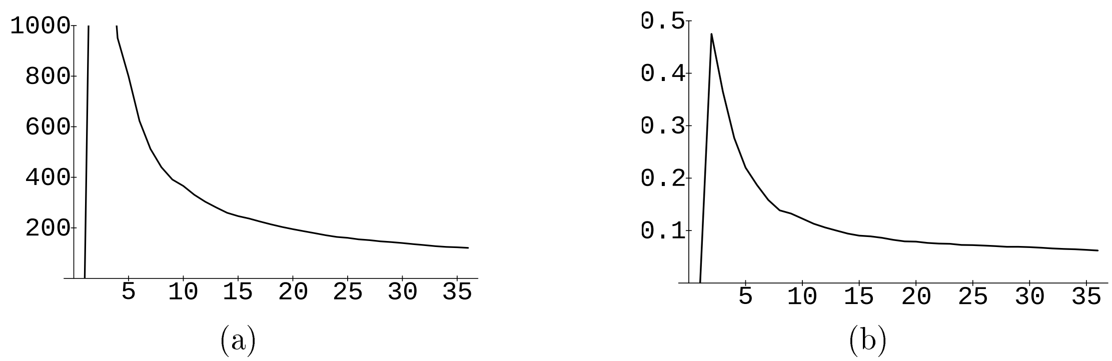


Figure 6.6: SpaceWorld error measures averaged over 10 350-visit runs. (a) Cumulative SSD/M. (b) Cumulative MAO/M.

similar to the learning component (excluding exploration and experimentation) of incremental active map-learners, such as [Kuipers and Byun, 1988]. The result of the normal run is shown in Figure 6.7(b); there are no surprises—the map converges relatively quickly to an accurate map, despite some ambiguities.

However, the naive system did not run quite so well, as is obvious in Figure 6.7(c). There are two main phenomena that gave the naive mapping system problems. First was the simple phenomenon of non-recognition of waypoints on subsequent visits. Since the system had no way of determining later that the waypoints were the same, no identification could be made. This led to proliferation of waypoint nodes (visible in the map shown in the figure) and an increase in geometric uncertainty. The second phenomenon that was noted was 'map crawling' as described above (but without possibility of correction). The effects of this crawling can be seen (with some difficulty) by the several door waypoint nodes in the constructed map. They spread out towards the right, due to misidentification of the two doorways during mapping. Figure 6.7(d) shows SSD/M for the naive mapping run. The first feature of the plot to note is that, at a large scale, it remains fairly constant throughout the run. That is, localization error remains high, since the map is constantly in a state of flux. At a finer scale, we see that the error alternates between brief rises and longer (but shallower) declines. Those rises correspond to periods after a critical misidentification when the robot's tracks fall out of synch with the map and the mapper starts mapping a portion of the

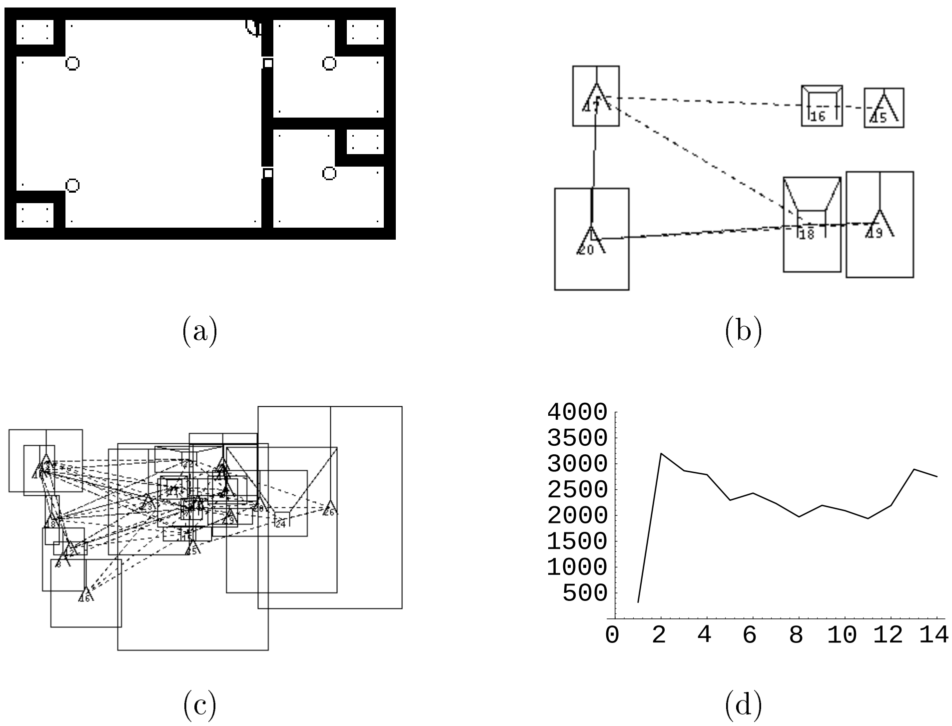


Figure 6.7: Comparison of map-learning results with and without error correction. (a) GreyWorld2 environment. (b) Example of a map correctly learned. (c) Example of a map 'learned' with error-correction turned off. (d) SSD/M without error-correction.

world over again. Once enough of the world is 'recreated', error starts falling slowly again.

In summary, we find that inclusion of geometric matching alone is insufficient to support reliable mapping. Use of error diagnosis and correction strategies appears to be necessary for accurate map learning.

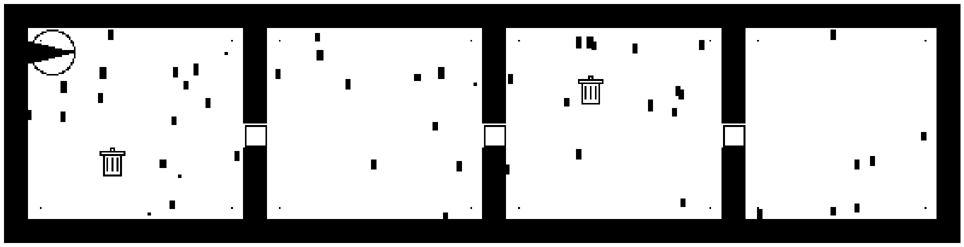


Figure 6.8: Initial state of the trash collection problem. Small filled rectangles denote trash, trash can icons denote trash cans.

6.5 Trash Collection

We also evaluated the use of the mapping system during execution of a realistic robot task. We used a “trash collection” domain similar to other pickup-and-delivery domains that have been investigated, both in situated robotics [Connell, 1989] and in planning [McDermott, 1992]. In the trash collection domain, the environment contains objects of two types: *trash* and *trash cans*[1]. Trash and trash cans are easily distinguishable from one another. Pieces of trash are of various sizes, and trash cans have limited capacity. Both sorts of objects are randomly placed throughout the environment, as depicted in Figure 6.8.

The robot does not start out knowing anything about either the environmental structure or object placement. Its task is to collect all the trash and deposit it in the trash cans. The reason we use this task is that there is a simple (though inefficient) default plan that requires no knowledge at all:

1. Wander until you find a piece of trash;
2. Pick it up;
3. Wander until you find a trash can;
4. Deposit the trash;
5. Go to Step 1.

[1] The environmentally conscious should read ‘recycling bins’.

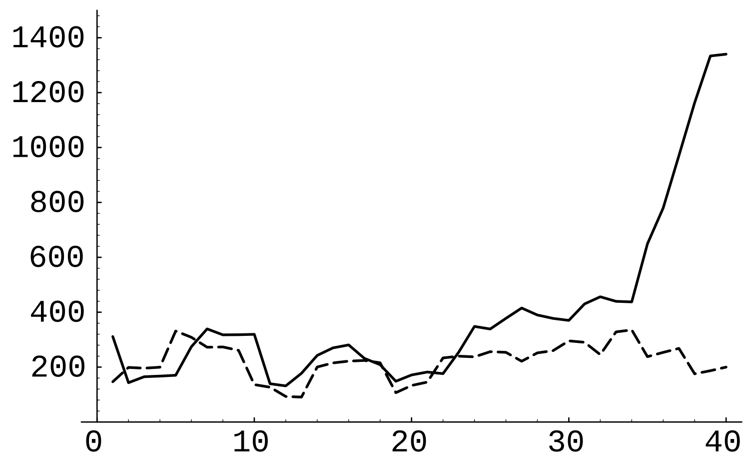


Figure 6.9: Trash collection results comparing with and without navigational planning. The x-axis shows the numbers of pieces of trash collected (40 in total) and the y-axis shows the (smoothed) time to deliver each piece (in simulator seconds). The solid curve is for random wandering, and the dashed curve is with map-based planning.

Provided wandering will eventually cover the entire world, this plan is guaranteed to work, eventually. (We ignore here the problem of determining when the robot is done and can go home.)

We enhance this basic plan to use navigation plans for finding trash cans, when possible. First, the robot notes trash cans in its map, treating them as a special type of waypoint (since they don't move). Then, every time a piece of trash is picked up, rather than wander randomly, the robot uses a navigational plan to head for the nearest trash can. If no trash can is known, the robot just wanders by default. Furthermore, whenever the robot is holding a piece of trash, it continually looks for a trash can. Thus, even if the robot is heading for a far-away trash can, it can take advantage of an opportunity if it finds another trash can along its route.

We compared the robot's collection performance in a world with 50 pieces of trash and two trash cans (see Figure 6.8), with and without planning. In both cases, the robot started with no map and built one along the way (though in the one case the map was ignored). A simple straight-line graph-search algorithm was used to generate plans, for simplicity. If no link existed between two waypoints, a heuristic algorithm based on those of Lumelsky and Stepanov [1987] was used to navigate to the approximate position of the destination waypoint; the waypoint was then sensed and approached. The robot was confirmed to be at a waypoint by the mapping system.

The use of the map was evaluated by measuring the time it took the robot to get piece of trash to a trash can. We recorded this time for the first 40 pieces of trash the robot collected (after that, it took far too long to wait for the robot to find the remaining pieces). The results of these tests are summarized in Figure 6.9. We smoothed the data using a window of five deliveries (in both runs, the robot occasionally had greater trouble finding a can, producing large spikes which obscure the structure of the data). When no planning was used, the time to collect trash starts out low, as the robot cleans up the areas near the trash cans. As the robot has to move further away to find trash, it takes much longer for it to randomly find its way back to the trash cans, and the time required rises precipitously. However, when the map is used to find the nearest trash can, the time it takes to get there remains about the same. Thus, even though executing navigational plans is somewhat more expensive per unit distance than random wandering (because of the extra sensing required), that cost is more than made up for by the fact that the robot is acting in a goal-directed fashion. In conclusion, we have demonstrated that the use of a passive mapping system during task execution significantly improves efficiency, even using a naive planning method.

Chapter 7

Opportunistic Exploration

We are confronted with
insurmountable opportunities.
—Walt Kelly, "Pogo"

Summary

In this chapter, we address the topic of intermittent exploration. The passive mapping paradigm requires that mapping not interfere with task accomplishment—this is done by making the mapping system independent of the robot's actions. However, exploration can still help the mapper, by focusing the robot's attention on portions of the world which are imperfectly mapped. Since the mapper does not require exploration for correctness, the decision about whether and when to explore can be left to the deliberator, which can trade off between exploration and task achievement. This idea has been embodied in the notion of *exploration scripts* which encapsulate short action sequences designed to gather information useful to the mapper in specific situations. If an exploration script is applicable in a particular situation, it is recommended to the deliberator, which can then decide to accept the recommendation or not. We show results which demonstrate how exploration scripts improve mapping effectiveness. One interesting phenomenon is the fact that having multiple scripts in the system is not necessarily an improvement, since they can interfere with one another. This necessitates estimating the relative utility of different scripts, so the best one can be chosen in a given situation.

7.1 Intermittent Exploration

In Chapter 5, we saw how passive mapping can be performed by using of explicit error correction. However, passive mapping can also be quite inefficient. There are two reasons for this. First, since the robot's movements are not determined by considerations of mapping, important parts of the environment may remain unexamined (or insufficiently examined) for long periods of time. More fundamentally, since mapping errors may persist for some time before they are noticed and corrected, construction of a correct map can be delayed. Matters can be improved by allowing the mapper control over the robot's behavior. If this control can be exercised intermittently when not interfering with the robot's other goals, the problems of purely active mapping can still be avoided. We call such action generically 'intermittent exploration'.

We conceive of intermittent exploration being implemented in an 'exploration adviser', which advises the deliberator when some exploration would be useful. The adviser examines the state of the map and the current tracks to determine an exploration plan for the deliberator to perform. We would like to be able to consider exploration strategies independent of mapping, so we suggest that exploration operate to improve mapping *indirectly*. By this we mean that exploratory actions move the robot so that the map is improved by the normal operation of the mapper; no information flows to the mapping system from the exploration plans themselves. Exploration will thus depend on the way the mapping system operates, but not vice versa. This approach is useful in that it allows exploration strategies to be developed modularly—the mapper can stay the same as new exploration methods are added. Such modularity is made possible by the passive mapping paradigm—the mapping system does not depend on exploration to work, though exploration may improve its performance.

7.1.1 Opportunity scripts

We propose here to use *opportunity scripts*, simple high-level robot programs, to improve mapping. Such scripts should be designed so that they can be applied whenever the robot decides that its high-level goals can be put off for a short time. In other words, scripts should not interfere greatly with execution of task-oriented plans. Since the underlying passive mapping system makes no assumptions about the actions the robot takes, scripts cannot cause much trouble for mapping.

The essential idea of using opportunity scripts to improve mapping is that they can be performed whenever a high-level decision process determines that other goals can be put off for a short while. This implies, first of all, that these scripts must apply in general circumstances, as the mapper has no control over when they will be called upon. Secondly, the scripts must be fairly limited in duration, so that they can do their business of improving mapping and then let the robot get back to its high-level goals. Scripts have two components, an application test which determines if the script is relevant, and a (loop-less) procedure which is executed if the script is applied. Scripts use the mapper's data structures, in particular the set of current tracks and the map itself. We discuss below a number of exploration scripts that we have implemented for use with our passive mapper, as well as the results we obtained with them.

7.1.2 Script utility

When should the deliberator decide to apply an opportunity script? If we suppose that the deliberator operates by choosing actions according to their expected utility, what is needed is some way of quantifying the utility of executing a script. The cost of executing a script can usually be estimated in the same way as the cost of navigational plans. The real question, then, is estimating the utility of the information the script will cause to be added to the map. This can be divided into three related problems. First is the question of *intra-task* utility of exploration. That is, how much does this exploration help me in the task I am trying to do now? Often the answer will be "not much", since presumably the deliberator has a good idea of how to achieve its current task, and if the script's actions were directly helpful the deliberator would have already considered them.

More significant is *inter-task* utility, the value of exploration to possible future robot tasks. There are two difficulties in formulating this problem. First is the fact that the robot must have some idea of what sorts of tasks it is likely to face in the future, in order to evaluate the usefulness of gathering information for those tasks. Second is the fact that we are not simply interested in the utility of performing a script versus not performing it at all; rather, we must consider also the possibility that the script could be performed at a later date (possibly when the robot has more leisure to explore). This requires evaluating the likelihood both of the script being applicable at a future time (possibly in the scope of a different task), and of the deliberator deciding to execute it then. This leads to a difficult recursive utility formulation.

Finally, we may consider the expected utility of executing a script to the mapping system itself. That is, how much is it likely to improve the map? This can be useful if the following decision heuristic is used by the deliberator: Execute an applicable exploration script if and only if its cost is below a threshold, ignoring its utility. The assumption is that the information gained will eventually be useful, so it makes sense to execute the script as long as it will not screw up the current task being executed. For example, if I need to be at a meeting in five minutes, I should not take time to explore an unknown side corridor; but if I'm in no hurry, I might as well explore a bit. Under this strategy, the utility of scripts need only be considered in order to decide among competing scripts if several are applicable at once. As the results of using exploration scripts shows (see below), this competition is empirically significant in terms of the gain of using exploration scripts. The problem is that the sorts of expected map improvements from executing scripts are different in kind, and are difficult to compare. Some scripts reduce positional uncertainty, others may discover new action links or remove false waypoints. Without a full theory of map utility (necessarily dependent on task assumptions, as discussed above), it is unclear how to properly formulate this problem.

7.2 Exploration scripts

An exploration script consists of a *test* and a *plan*. The test is applied to a current robot state (the current map and a track) to determine if the script is likely to improve the map. If so, the plan is executed, possibly with some variables instantiated by the test (for example, a waypoint that should be visited). Since the utility problem is difficult, as discussed above, it is not addressed in this implementation; scripts are applied some fraction of the time when applicable.

We have investigated a number of exploration scripts. Some seek to reduce positional uncertainty. Others attempt to find new waypoints and action links. Scripts are also used to reduce ambiguity in the map or in the robot's position estimate. Some probe map waypoints which seem likely to not really exist. Keep in mind that all these scripts do is direct the behavior of the robot; they direct the attention of mapping system, improving the robot's map. Strategies similar to some of these scripts can be found in the robot mapping literature; the main contribution here is the uniform framework in which they are embedded.

7.2.1 Expansion

One of the simplest ways to improve a map is to extend its coverage to previously unknown areas. The following script does just that.

<u>HEAD FOR UNEXPLORED AREA</u>:

This script attempts to head for unknown areas ripe for exploration in order to improve the coverage of the robot's map. To determine if an area near a known waypoint is unexplored, we use the traversal confidence grid described in Section 5.5.3. To briefly summarize, the confidence grid estimates the likelihood $t(x, y)$ of the robot having ever moved across point (x, y). We can easily compute a function $T(\mathbf{x})$ which returns the confidence of the robot having ever traversed some part of the area represented by the interval $\mathbf{x}$, as $T(\mathbf{x}) = \max_{(x,y)\in\mathbf{x}} t(x, y)$. An area $\mathbf{x}$ is deemed to be unexplored if the likelihood of part of it having been visited in the past is sufficiently low, $T(\mathbf{x}) < \vartheta_E$[1]. Thus, HEAD FOR UNEXPLORED AREA looks in the vicinity of the robot's current waypoint for a nearby area which looks unexplored, and if one is found, attempts to head in its direction.

[1] Thresholds are notated as ϑ_i, where i is a unique identifier for the threshold.

If:	• Uncertainty of the last Δposition high • Uncertainty of all track positions high • Didn't just retrace step
Do:	• Try to get a fixation on the last waypoint visited; • If found, go there, otherwise exit; • Try to return, if possible.
	RETRACE STEP

If:	• There is a nearby waypoint whose uncertainty is high • Only one track is current
Do:	• Try to get a fixation on the uncertain waypoint based on its relative position; • If fixation found, go there, otherwise exit.
	HEAD FOR UNCERTAINTY

If:	• There is only one track, with high uncertainty • There is nearby waypoint with low uncertainty
Do:	• Try to go to that waypoint.
	HEAD FOR CERTAINTY

If:	• There are multiple current tracks.
Do:	• Choose an action link whose destination is reachable from some, but not all, of the tracks' waypoints; • Try to go to that link's destination waypoint.
	DISAMBIGUATE TRACKS

Figure 7.1: Summary of refinement scripts

7.2.2 Refinement

Another class of exploration scripts are those which attempt to refine information already in the map or in the robot's location estimate. Those described below generally try to reduce positional uncertainty in the map or in robot tracks.

RETRACE STEP:

One important source of error and ambiguity in a map is positional uncertainty. When the robot performs an action with a very uncertain estimate of relative position, $\mathrm{rad}(\mathbf{x}_o) > \vartheta_{RSo}$, and the robot's *a posteriori* position estimate (after matching to its map) in all tracks is also particularly uncertain, $\mathrm{rad}(\mathbf{x}_t) > \vartheta_{RSt}$, a simple and useful heuristic for reducing uncertainty both in the map and in the current position estimate is to retrace the last step taken. That is, the robot tries to return to the waypoint it just came from; if it manages that, it tries to get back to where it started.

HEAD FOR UNCERTAINTY:

A more generally applicable heuristic for reducing map position uncertainty is simply to head for a nearby waypoint with large positional uncertainty, $\mathrm{rad}(\mathbf{x}_w) > \vartheta_{HU}$, assuming that if it is reached, new constraints will improve the positional estimate. This can only be reasonably tried, of course, if the robot's current location is unambiguous.

HEAD FOR CERTAINTY:

The converse of the last script is useful when the robot's positional uncertainty gets too high, $\mathrm{rad}(\mathbf{x}_t) > \vartheta^t_{HC}$. This strategy is to head for a nearby waypoint whose position is known fairly precisely, $\mathrm{rad}(\mathbf{x}_w) < \vartheta^w_{HC}$. If the robot reaches and recognizes the waypoint, uncertainty in the robot's position will be reduced, improving further localization and mapping.

DISAMBIGUATE TRACKS:

It will often occur that the robot's estimate of its current location will be ambiguous, resulting in multiple tracks. If this situation obtains, a good way to distinguish between them is to try to perform an action with different results for the different

If:	• There is a single current track, • There are multiple links from the current waypoint labeled with the same action.
Do:	• Perform that action.
	PROBE AMBIGUOUS ACTION

If:	• There is a single current track, • The current waypoint was recently split off from another waypoint.
Do:	• Choose an action link which both waypoints have in common, • Try to traverse it.
	PROBE SPLITTAGE

If:	• There is only one current track. • The current waypoint has a neighbor which has been visited less than a threshold number of times.
Do:	• Choose such a rarely visited neighbor, • Try to go to there.
	HEAD FOR RARE WAYPOINTS

Figure 7.2: Summary of confirmation scripts

possible robot waypoints implied by the different tracks. We do this by choosing a link for some tracked waypoint w_{t_1}, whose destination waypoint is not the destination of any link for another tracked waypoint w_{t_2}. If the link is traversed successfully, the inconsistent track will probably evaporate.

7.2.3 Confirmation scripts

The last type of exploration scripts that we will discuss here are ones which attempt to reduce ambiguity in the map by trying to disconfirm specific weak hypotheses about possible mapping errors. The hypotheses tested here are about map features that seem likely to be artifactual. The scripts thus probe them to gather evidence which will be used either to strengthen the features or remove them.

<u>PROBE AMBIGUOUS ACTION</u>:

One kind of map ambiguity is when a waypoint has multiple action links from it (to different destinations), labeled with similar actions[2]. While this ambiguity may be inherent, it may also be that one of the links is due to a transient; hence, further examination is warranted. This is achieved by attempting to perform such an ambiguous action—this will tend to speed up elision of transients, and just maintain the real action links.

<u>PROBE SPLITTAGE</u>:

The kind of map ambiguity we consider for now is where two identical links from different waypoints end at the same waypoint and there is reason to believe that only one is real. One situation in which this supposition makes sense is when the waypoints in question resulted from a waypoint splittage (see Chapter 5). Since both waypoints resulting from a split have copies of the same action links (or some of them, at least), many of those links will be invalid. Hence, when the robot is at a waypoint which resulted from a recent split, the PROBE SPLITTAGE script tries to perform an action that has (in the map) identical consequences in the two split-off waypoints. This accelerates elision of the invalid action links.

[2]This may seem unlikely, *a priori*, but occurs commonly for two reasons. First of all, before the map is complete enough to be useful, a generic wandering routine is often useful for default task performance. Such strategies tend to repeat their behavior in similar situations. Second, once the map is used for navigation, action link plans are used as subplans. Thus one action link's action may be 'duplicated' by being executed as part of a larger navigational plan.

Head for rare waypoints:

Recall that transient environmental features are eventually removed by the mapping system by noting their frequency of apprehension. This process can be speeded up if the robot tries to reach waypoints which look likely to be transients. Since transients, by their very nature, are not encountered often, it is likely that a waypoint that has not been visited often is transient. If so, then repeatedly trying to reach the waypoint will cause the system to notice that it is not encountered as expected, and so it will eventually be removed. If it is not a transient, then the mapper will gain more information about the waypoint, which is also useful.

7.3 Results

To determine the advantage of using exploration scripts, we first ran experiments in the simulated world shown in Figure 7.3. (Please refer to Section 6.1 for an explanation of our evaluation methodology.) The robot moved on a random walk, when not controlled by exploration scripts. Experiments were run for different sets of exploration scripts active. Each experiment consisted of 10 mapping runs, with the robot making 400 moves (from place to place) in each run. Cumulative SSD was sampled each 10 moves. Active exploration scripts were applied 50% of the time, when applicable. The results are summarized in Figure 7.4. As can be seen from Figure 7.4(a), application of any one exploration script provides a noticeable advantage over a random walk. The mapping improvement provided by any particular script depends, of course, on the structure of the world. Figure 7.4(b) shows the effect of using exploration scripts together and shows an averaging effect; this is due to interference between multiple applicable scripts as discussed above.

Another experiment was run testing the use of exploration scripts for efficient map learning. This was on the large and complicated environment shown in Figure 7.5. Intermittent use of exploration scripts (applied 10% of the time, when applicable) allowed this world to be accurately and almost completely learned in less than 3000 waypoint visits. This is one of the most complex maps learned by any mapping system similar to ours. Note that very little controlled action was used to achieve this result.

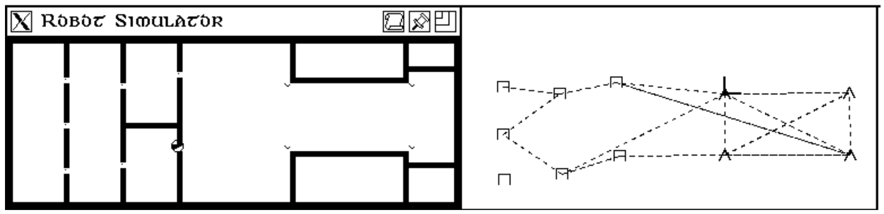


Figure 7.3: The MultiHall environment and a typical map learned.

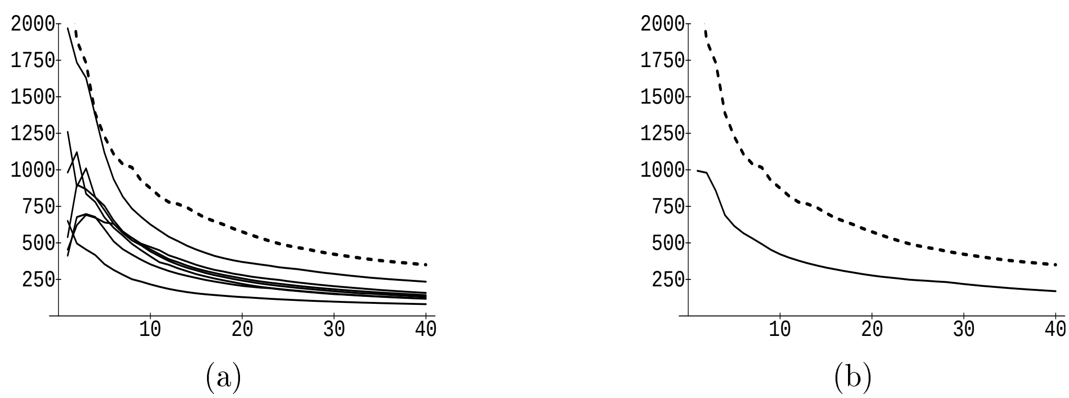
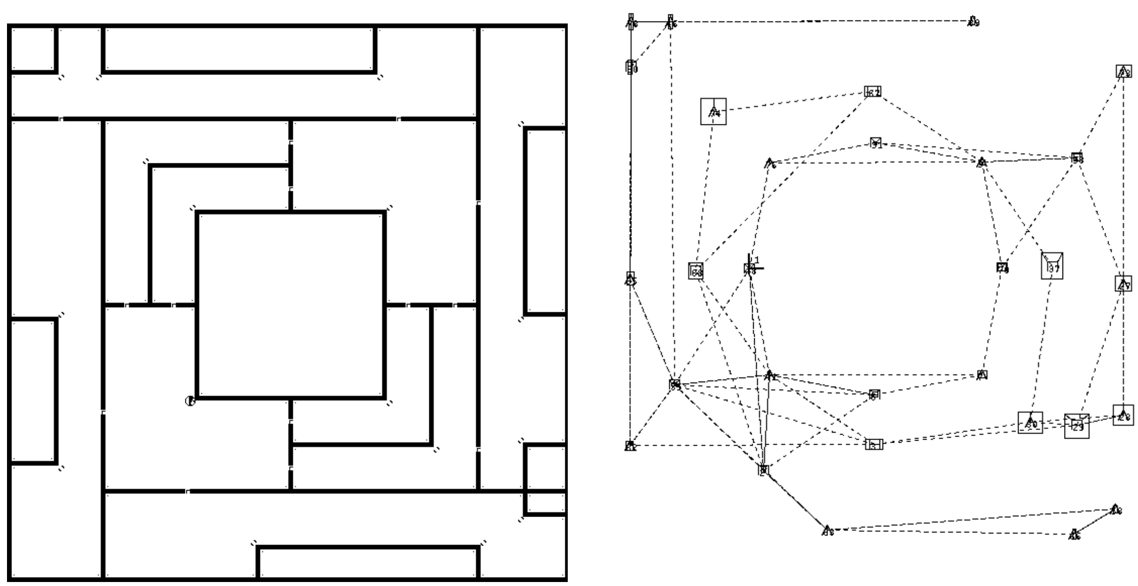


Figure 7.4: Exploration script performance: SSD/M v. number of moves. (a) With no exploration scripts (dashed) and single exploration scripts (solid). Note that all scripts provide significantly reduced prediction error. (b) No scripts (dashed) v. all scripts (solid).



(a) (b)

Figure 7.5: A complex environment learned with the help of exploration scripts.

Chapter 8

Related Work

In this chapter, we will survey the research on mobile robot mapping and estimation. Since most of such work involves large and complicated systems, space does not permit a full exposition. Hence we limit our view to those aspects of the literature that are both relevant to the map learner described in this dissertation, and that are specific to each particular system. This chapter is organized into a semblance of a classification of the related work, but since each piece of research addresses many issues at once, any such taxonomy can only be approximate. In some cases different aspects of a single system are discussed in different places when the exposition is clarified thereby.

8.1 Qualitative representation

Topological methods represent qualitative relationships between environmental features (landmarks, places, etc.), and so can be more robust than precise geometric modeling, with respect to both sensor error and environmental change. For this reason, various topological mapping schemes have been used in robotic research. Detailed geometric modeling is not necessary, since local navigation can usually be performed using only sensor feedback. However, as noted in previous chapters, purely topological representations cannot, in general, be learned passively (without exploration), so we have chosen to incorporate metric information into our maps. Some of the systems described in this section do so as well; however, except for Levitt *et al.*, all rely on active exploration strategies in order to build their maps.

8.1.1 Path-based representations

The first broad class of qualitative representations are those based on some sort of path graph. There are two defining features of path graph representations: paths are primitive, and represented locations are point-like. By paths being primitive, we mean that representations are built based on primitives which represent the corridors of the robot's movement through space. This may be contrasted with other approaches, in which paths are built out of set of adjacent regions of freespace (see below). The locations that are represented by nodes in a path graph are considered to be on the order of the robot's size so that they both constitute useful nexus points between paths, and afford relatively accurate robot localization[1]. Within these constraints, a number of different representations have been developed, incorporating a variety of topological and geometric techniques for dealing with place recognition and navigation.

TOUR

Probably the earliest comprehensive path-based map representation scheme was that in Kuipers' TOUR model [Kuipers, 1978]. TOUR was developed primarily as a model of human cognitive mapping. It uses a relatively simple world model, based on human sensorimotor capabilities in path-structured environments (such as buildings or city streets). Places are identified with views, and the agent can either turn in place or move in straight lines. Sensing and movement are assumed to be noiseless. There are two primary components to TOUR's qualitative map: a procedural representation of route knowledge, and a graph-based representation of topological relations between places. The procedural representation supports map-less navigation by using a set of schemata, each of which specifies what action should be taken (and result expected) for a given combination of current view and goal. When enough schemata have been acquired, the system integrates them into a topological map. In addition to the

[1] Though useful, this constraint is not strictly necessary; Mataric [1990] uses a sort of behavioral representation to good effect here (see the discussion below).

usual information about paths between places, TOUR's map includes containment and boundary information relating paths and places. Computing this information, however, requires strong assumptions about lack of sensor noise and action determinism, so the other systems described below have not been able to use such information. TOUR also includes some metric representation, but it assumes essentially no metric uncertainty.

Kuipers & Byun

While it constituted a large step towards a computational understanding of human map-learning and navigation, the TOUR model made strong assumptions about sensing and action that are not valid for real robotic systems. Therefore, Kuipers and Byun [1988] apply the path-related ideas of the TOUR model to a simulated robot, called NX. The key notion in NX's mapper is the idea of the Distinctive Place (DP), a local maximum of one of several sonar-based 'distinctiveness measures'. DPs are a sort of waypoint, as we discussed in Chapter 4. The primary structure in the NX map is a graph whose nodes represent DPs, and whose arcs represent the paths between them. Defining DPs as local maxima means that the robot can actively localize itself, by hillclimbing on a relevant distinctiveness measure. DP nodes are annotated with the sonar readings gotten at the location, which are used as views for perceptual matching. Mapping is performed with the robot controlled by an exploration strategy designed to get the robot to move towards unknown areas of the environment as often as possible. Furthermore, whenever local sensor readings do not suffice to identify the current DP, the robot executes a 'distinguishing sequence' of actions designed to disambiguate the different possible DP nodes it may be at. The NX mapper was the starting point for the work described in this dissertation—the idea was to develop a passive path-based mapping system.

CARMEL

A approach similar to Kuipers and Byun is taken by Kortenkamp *et al.* [1992] in their work on the University of Michigan robot, CARMEL. They use a similar path graph, whose nodes represent *gateways*, rather than DPs. Gateways are also a class of waypoint—they represent transitions between different spaces in the environment.

The idea is, however, that they constitute a more natural basis for mapping than arbitrary DPs. Local visual sensing is used to characterize gateway views—vertical edge images are used and matched to one another for gateway identification and robot orientation. Vague geometrical information about the relations between gateways is attached to gateway nodes, as described further below. The mapping process at present relies on a man-made exploration path, which takes the robot repeatedly along several routes in the environment. The experiments also were carried out over small areas and it is not clear how the results would scale up.

LOGnets

Another graph-based representation scheme with some interesting features is the LOGnet representation of Malkin and Addanki [1992]. Nodes in their graph represent ‘distinctive locations’ which are defined as places the robot encountered an obstacle. This is only reasonable for environments containing few movable objects, but this definition of place is not critical to the rest of the approach. One important aspect of the approach is the use of action sequences (rather than atomic labels) for labeling the arcs of the LOGnet. The action primitives are, however, very low-level primitives, such as `turn-left` or `go-forward`, which are executed at about 1Hz. This approach, therefore, lacks robustness in the face of environmental changes (such as moving obstacles), and also does not produce useful information for a deliberator to use in constructing reactive plans. Our approach to behavior abstraction, described in Section 4.5, remedies these problems by representing behavior as reactive plans. Another interesting feature of the LOGnet approach is the use of polar certainty grids to represent locations. These grids (described below) represent the geometric structure of the environment in the vicinity of represented locations; grids can be compared in order to verify the robot’s location.

Toto

In her master’s thesis, Mataric [1990] proposes a novel path-based map representation for her robot Toto. The representation scheme is a path-graph whose nodes represent ‘landmarks’. A landmark, to her system, is a portion of the world with a reliable relationship to the robot's behavior[2]. For example, an instance of the robot following a wall on its left (for a long enough time) constitutes a landmark. Although a landmark (i.e, a place) in Mataric's system may occupy a large portion of space, they still operate as path-based nodes. Since Mataric's landmarks are behaviorally stable portions of robot paths (and not arbitrary regions of space), they can serve as path nexi just like point-like nodes (such as waypoints). Toto's map construction algorithm relies both on a specific hard-wired behavior pattern ("follow object boundaries") and on some simple landmark identification heuristics. Two landmarks are assumed to match if their types (eg, "wall on left"), approximate positions, and approximate compass headings (eg, "robot heading north") match. In addition, expected landmarks were preferred. These simple strategies suffice because of the robot's fixed behavior and the small size of the environments learned. Although Toto's mapper does not use active exploration, the success of the mapper appears to depend crucially upon the fact that Toto's behavior is highly structured.

8.1.2 Abstract models

FSM inference

Formal work on map-learning inspired problems has focused on essentially path-based formalisms, modeled as finite state machines (FSMs) of various sorts. Places are modeled by FSM states, actions by transition labels, and percepts by state outputs. The map learning problem is then formulated as the problem of inferring, from known transitions and observations of state outputs, the smallest FSM consistent with the data. Since passive FSM inference is known to be intractable [Angluin, 1978,Gold, 1978,Pitt and Warmuth, 1989], research has focused on how to effectively use exploration to infer FSMs. Rivest and Schapire [1989] describe an algorithm for learning arbitrary finite state machines using homing sequences.

In order to reasonably represent real robot behavior, however, FSM models must incorporate sensor and effector uncertainty. Dean's group at Brown University has provided some important results in this area. They have shown that even if state transitions have probabilistic errors, but the correct transition occurs more than half

[2]This usage of the term 'landmark' is somewhat nonstandard; it should not be confused with the use of the term elsewhere.

the time, maps can often be learned in polynomial time with exploration [Basye *et al.*, 1989] (using Rivest and Sloan's [1988] *probably almost always useful* learning criterion). In particular, they present algorithms for learning maps if either:

- The environment is a regular tessellation, or
- The robot always knows the action associated with its last transition (*reverse movement certainty*).

This work assumed that state outputs are perceived correctly; more recent work has started relaxing this requirement. Dean *et al.* [1992] have also shown how to PAC-learn a finite state machine with up to 50% sensing error of state outputs, provided the actions are deterministic and the machine has a (known) distinguishing sequence.

Predictive distinctions

Another approach to learning, one with some close connections to this dissertation, is Chrisman's method of *predictive distinctions* [Chrisman, 1992]. His focus was not explicitly on the map learning problem, though; the problem he addressed was that of *perceptual aliasing* in reinforcement learning. Reinforcement learning is a class of techniques for learning action-selection policies which maximize overall performance (see, eg, [Sutton, 1984]). Explicit rewards from the environment are used to infer an 'optimal' control policy, which usually takes the form of a function from percepts to actions. This approach can be problematic when important distinctions between world states are not captured by direct perception (perceptual aliasing). Perceptual aliasing is, of course, one of the main reasons for building internal representations of the world—Chrisman's work extends the reinforcement learning paradigm in just this way: an internal representation is learned, so that policies are functions of inferred states rather than direct percepts. This allows context to help distinguish between different states which look similar.

Chrisman models the world essentially as a non-deterministic FSM, with stochastic transitions and state outputs. The model includes estimates of transition and output probabilities. The robot's belief about its current state is a vector representing the probability of the robot being in each state. When the robot performs an action, its belief is updated based on the action performed and output received, using Bayes' rule. Given a state model, a variant of Q-learning [Watkins, 1989] is used to learn a policy relating model states to desired actions. Model learning is interleaved with policy learning. The system is initialized with a small random model. As the system operates and learns a policy for the model, it remembers all its actions and outputs. Every so often (in Chrisman's tests, every 1000 actions) these stored memories are used to improve the model. There are two sorts of model learning, corresponding to 'update' and 'restructuring' in the ASA (see Section 5.1.1). First, the system uses the observed output frequencies to update the transition and output probabilities of the model. Second, statistical comparisons are made on different parts of the accumulated data in order to determine if a model state leads to different predictions under different policies (this is possible without explicit experimentation since the policy is constantly changing during operation). If this is the case, the state in question is split into two. In this way, state distinctions which do not depend directly on perceptual distinction can be discovered. This method is similar to the connective method for diagnosing monotopy described in Chapter 5. There are, however, two differences in our assumptions. We can use geometric constraints for teasing confused states (waypoints) apart, while Chrisman cannot. Conversely, he can use his knowledge that the reinforcement learner is slowly changing a policy for a fixed goal so that he can use a simple statistical test to detect inconsistency; we cannot make such strong assumptions about the underlying action-selection policy.

8.1.3 Regions

Instead of explicitly representing robot paths, some researchers have chosen to represent relationships between different (large) regions of space. This typically means representing the world as a graph whose nodes represent regions of space, with links between adjacent regions. We can call such a graph a *region graph*. Two approaches are taken to constructing such a graph, direct and indirect. The direct approach attempts to infer large-scale region topology directly from sensing, without building a geometric model of the environment. The indirect method first builds a geometric model of the environment then divides it into regions from which a topological description of the environment can be extracted. The indirect approach is often easier, since the geometric representation that is constructed allows a global view when constructing topological representations. However, the approach is more expensive (since a detailed geometric model must be made), and can be overly sensitive to errors in the geometric model.

Adit networks

Crowley [1985] develops a region-based representation scheme for use in his IMP (intelligent mobile platform) architecture. The robot operates in an indoor environment. It first builds a polygonal representation of free space (reduced by the size of the robot so it may be treated as a point) by fitting lines to sonar readings to estimate wall positions. The free space is then subdivided into convex regions which serve as the basis for the topological representation. In addition to using the region graph for coarse path-planning, its dual is also used, where nodes represent *adits*, defined as transitions between adjacent regions. All adits adjacent to a region are linked together; since regions are convex, there is a straight-line path between any two linked adits. This allows detailed trajectories to be easily computed, given region-to-region paths. The adit network constitutes a path graph, however, unlike our approach (and those described above) the geometric representation is taken as primitive, with paths being inferred from geometric relationships. This has several undesirable effects. Since the method relies on convex decomposition of a geometric model, it is inherently inefficient (since a complete geometric model must be built first). Also, the path representation is sensitive both to sensing errors (sonar specularities, for example) and small environmental changes (such as moving chairs or tables).

Yeap

Yeap [1988] describes a cognitive mapping system developed as a computer model of human map-learning and navigation capabilities. As an agent moves around, it extracts local models of its current region, assuming straight walls. Places are represented as polygonal spaces, derived using an occlusion-based heuristic from sets of perceived wall segments. Adjacent places are linked, provided the adjacency permits movement between the two places; place regions may overlap. The place network is represented purely relationally, no metric information is used. Place polygons are used for recognition and navigation, and are represented metrically; noiseless sensing is assumed, so spatial uncertainty is not modeled.

QualNav

Levitt *et al.* [1987] describe QualNav, a direct region-based map representation scheme for outdoor navigation. The basis of their method is the notion of the *landmark pair boundary* (LPB), which is a line joining a pair of identifiable landmarks (such as buildings or mountains). The set of all LPBs divides the world into a set of convex regions, each of which is uniquely identified by its relationship to adjacent LPBs. These regions for a hierarchy, with more inclusive regions being defined by fewer LPBs. If the robot identifies two landmarks, it can determine which side of the associated LPB it is on, by the relative positions of the landmarks (which one is on the left, say). When a set of landmarks has been identified, the robot can localize itself more precisely, and determine which region it is in. Path planning between regions involves identifying a set of desired LPB crossings; these can be performed by using visual feedback on tracked landmarks. This method avoids the problems of indirect region representation (as above) since no explicit geometric model needs to be constructed. Plan steps (i.e, LPB crossings) constrain the robot's behavior only a little, so they allow for great flexibility in execution, which is also one of the hallmarks of the path-based approach. However, using LPB maps requires that much of the environment be visible at any given time so that LPBs between distance landmarks can be sensed.

8.2 Metric representation

This section surveys work in geometric representation related to the issues addressed in this chapter. Such work includes set-based and statistical representation methods used in mobile robotics, as well as work on grid-based representation methods. There is a considerable literature on statistical methods for estimation and control which we will not review here, as it is peripheral to the topics of this chapter. The interested reader is referred to [Gelb, 1974].

8.2.1 Sets for estimation and control

The use of bounded uncertainty representations like that we use has a long, though sparse, history in the estimation and control literature. Schweppe [1968] describes a set-based estimation method for continuous linear systems using bounding ellipses. This method was extended and improved by Bertsekas and Rhodes [1971], who also noted the close relationship between ellipsoidal estimation and Kalman filtering for particular classes of noise models. At the same time as Schweppe, Witsenhausen developed set-based methods for discrete-time linear systems using support functions for convex sets [Witsenhausen, 1968b]. He also relates this problem to that of minimax control [Witsenhausen, 1968a]. Barmish and Sankaran [1979] analyze the use of polytopic representations of sets for modeling uncertainty in discrete linear systems; this idea was also suggested by Witsenhausen [1968b].

In [Hager *et al.*, 1993], we compared set-based estimation methods with comparable statistical methods, with particular attention to robotic applications. We concluded that the two methods are generally appropriate for two different domains. Set-based methods typically perform well in cases where the observation system is invertible and errors are bounded and not strongly centralized. Efficient implementation may be more difficult in higher dimensional spaces, however. Statistical techniques work best for nearly linear systems with statistically well-characterized errors, and particularly when errors are strongly centralized. Both methods can usually be adapted to cases where their assumptions are weakly violated; of course, there are situations that are difficult to deal with using any method. We also noted that applications involving hypothesis testing may benefit more from set-based methods than similar applications requiring accurate point estimates.

8.2.2 Set-based representation

SPAM

McDermott and Davis [1984] describe a system for general spatial reasoning, called SPAM. It supports many forms of geometric and topological reasoning, but the only aspect of SPAM relevant here is its representation of spatial relations and uncertainty.

SPAM represents several sorts of spatial parameters, including relative position, orientation, and scale. Spatial parameters are represented with respect to local reference frames, called *frobs*. Uncertainty is modeled using *fuzzboxes*, which are intervals in the several spatial dimensions, similar to the interval representation discussed above in Section 4.4.5. Due to the complexity of exact estimation of spatial relationships in this framework, SPAM uses two approximate techniques: a Monte Carlo method and conjugate-gradient hillclimbing. Our formulation in Section 4.4.4 avoids these difficulties by using path consistency as a weaker criterion for estimation correctness.

MERCATOR

MERCATOR [Davis, 1984] is a mapping system which represents objects as polygons. Spatial uncertainty is represented in two ways: *grain-size* and *fuzz*. Grain-size is used to parameterize uncertainty in the shape of objects approximated by polygons. Fuzz, as above, connotes intervals on spatial relationships. Some of the relations between polygon vertices are represented in MERCATOR maps. Performing exact relationship retrieval in Davis' formulation is computationally intractable, so he uses a Monte Carlo technique (as in SPAM) for estimating geometric relationships between vertices. Learning is formulated as *map assimilation*. The robot is assumed to perceive a partial map, which is matched to the robot's existing map and relationships are updated. Fuzz is easily updated by interval intersection. More complicated is the combination of different polygons approximating the same object. This is done by searching for a polygon on the union of both inputs' vertices which minimizes a heuristic 'weakness' based on total grain-size, fuzz, and perimeter.

Atiya & Hager

Atiya and Hager [1991] describe a system for mobile robot localization using a set-based approach to stereo vision. They triangulate features (vertical lines) to find their 2D positions. Positional uncertainty (due to pixel quantization) is represented using intervals. They note that time-series analysis of stereo estimation error did not have any simple statistical characterization, but errors were well-bounded. Therefore, positional uncertainty (due to pixel quantization) was represented using intervals. Localization was performed using an a priori model of feature positions. Correspondence between scene points and model points was determined by matching invariants, using intervals to describe parameter uncertainty. The maximal consistent correspondences can then be used to determine the robot's absolute location (within some interval). It turns out that for this problem, the set-based method performs significantly better than the equivalent statistical formulation [Hager *et al.*, 1993].

8.2.3 Statistical techniques

Stochastic maps

A general statistical method for metric representation is described by Smith, Self, and Cheeseman [1986]. They formulate the notion of a *stochastic map*, which is a composite estimate of the positions of observed with uncertainty represented by the covariance matrix of all the point parameters. In our framework, a stochasting map is a complete reference graph, where links are labeled by the mean and covariance of the relative position of their endpoints. An Extended Kalman Filter (EKF) is used to incorporate new observations into the stochastic map. This amounts to a tacit assumption of Gaussian noise, which is a dubious assumption for odometric errors (and for visual errors; see [Hager *et al.*, 1993]). In addition, the linearization required by the Extended Kalman Filter has problems with large angular uncertainty (they require noise be less than 5 degrees). One useful property of their technique, however, is the ability to easily incorporate into the map general geometric constraints between sets of points by using linear approximations.

Hilare

The geometrical modeling system of the Hilare project [Chatila and Laumond, 1985] represents objects and spaces in the environment by convex polygons. Each polygon's shape is represented in a local frame of reference, in order to deal with the referential locality problem. These object-local frames are, however, related to each other in a global frame of reference. Uncertainty is represented statistically using Gaussian distributions and combined using an EKF. Polygons are linked together in a graph structure for route planning (similar in style to Crowley's adit networks, see above).

Leonard, Durrant-Whyte, & Cox

Leonard, Durrant-Whyte, and Cox [1992] present an EKF-based mapping system using sonar-derived features. There are two primary innovations in this work designed to deal properly with the problems of sonar sensing. As they note, the naive use of sonar for depth measurements in highly unreliable, since sonar tends to reflect specularly from walls, so multiple reflections are common and range readings are invalid. First was their use of *regions of constant depth* (RCDs) for extracting reliable features from sonar. Building on the realistic sonar models developed in [Kuc and Viard, 1991], they note that when nearby range estimates are similar, the estimates tend to be highly accurate. They further note that if two range readings from slightly different positions are used, the robot can distinguish between RCDs due to planar surfaces and those due to corners. These constitute the two main feature types in their mapping system. They further deal generally with the problem of outliers by associating each feature with a confidence, which is increased as the feature is repeatedly seen, and is decreased when the feature is predicted but not seen. When a feature's confidence drops below a threshold, it is forgotten. This is a general technique that we also use to deal with outlying observations. They determine the correspondence between map features and observations using a Mahalanobis distance test. However, they do not provide for explicit correction of mapping errors, tacitly assuming that no false matches are made in correspondence. They show good results for a small environment with no odometric noise.

8.2.4 Grid-based techniques

A different philosophy of spatial representation is espoused by those using *grid-based representations*. These methods, rather than using spatial relationships as primitives, represent an entire space by a grid tessellating the set of possible parameter values. Each grid cell is then associated with one or more features whose configurations fall within those denoted by the cell. Uncertainty can be represented by indexing features in multiple cells and by associating confidence levels with feature placements. Grid representations are easy to implement, but retrieval and matching operations can be expensive due to the cost of examining the entire grid. Hence grids are less useful for domains with sparse feature sets. Also, the resolution of the system is limited by the cell size, which is not the case with the direct methods discussed above.

Certainty grids

The certainty grid of Moravec and Elfes [1985] is the prototypical grid-based spatial representation. The environment is represented by a grid in which each cell represents the occupancy of its corresponding part of the world. Each cell contains an estimated probability that a part of the world is obstructed by some obstacle. Cell probabilities are updated from sensor readings using empirical sensor models and Bayesian updating. Odometric noise is accounted for by convolving the certainty grid with a Gaussian mask, to account for loss of precise registration. To compensate for this monotonic loss of information as the robot moves, a registration procedure is introduced in [Elfes, 1987], however I am unaware of any analysis of the correlation that this process introduces into the certainty grid. In any case, old observations tend to get smoothed out beyond usefulness after a short while, so certainty grids are most useful for local maps. We use a coarse certainty grid approach to modeling the robot's knowledge about the world for deciding among competing hypotheses, as described in Section 5.5.3. Certainty grids were also used by Taylor and Kriegman [1993] for the construction of local maps. They connect the grids in a graph, whose links are labeled with approximate geometric information (hence is a reference graph).

Regional maps

In their otherwise topological mapping framework, Kortenkamp *et al.* [1992] introduce *regional maps* for representing rough large-scale metric information. Associated with each waypoint a map (which they call *gateways*) is a set of polar-coordinate grids, each of which represents a large region of space adjacent to the waypoint. Each grid cell contains pointers to other waypoints whose positions are determined to fall within the cell. This representation is used for large-scale route planning, and is largely independent of the topological mapping process used in their system. This bears some similarity to our visitation confidence grid described in Section 5.5.3, except that their regional maps contain pointers to particular waypoint nodes.

Chapter 9

Discussion

"The time has come," the Walrus said,
"To talk of many things—
Of shoes, and ships, and sealing-wax,
Of cabbages, and kings,
And why the sea is boiling hot,
And whether pigs have wings."
—Lewis Carroll

In this dissertation, we have introduced and developed a paradigm of passive robot map-learning. The ultimate goal of this research is to develop robust map-learning methods that could be applied to enable mobile robots to operate intelligently in open-ended environments. Several important contributions to this long-term project have been described here. An abstract robot domain has been developed, justified by robotics research. It has been argued that its level of abstraction is appropriate for research into high-level learning and planning, and the simulator provides flexible facilities for experimental design. The core of the work was the development of the passive mapping system (implemented in the simulator), which operates by properly addressing localization ambiguity and by incrementally correcting learning errors. The representation used by the mapping system is tailored to support robust navigation and transformational planning. In addition, intermittent exploration can easily be incorporated to improve mapping efficiency, as shown in Chapter 7. Visual place recognition for mapping is addressed through an effective image-based matching technique using image signatures.

The robotic domain developed for this dissertation was designed with an eye to the two complementary problems of AI domain design: *abstraction* and *relevance*. A domain must be abstract enough so that the researcher's attention is not diverted from the issues of interest, but at the same time there must be assurance that the solutions in the abstract domain are relevant to the real world. Relevance is obtained by justifying the abstractions used and verifying proposed solutions in the real world. The robot domain that was developed incorporates realistic abstractions of object sensing, visual place recognition, and behavior-based robot control. The abstractions that were used in the robot domain embodied in the Ars Magna simulator are justified current research in robotic perception and control.

One important problem in map-learning is place recognition. Chapter 3 described a technique for using visual perception for place recognition, using image signatures. The main idea is to take advantage of the constraints on the place recognition problem provided by the topological mapping framework. The primary assumptions are that positions of interest to the robot are sparse (waypoints) and that the robot can reliably find them by low-level control. These assumptions enable us to apply an efficient image-based matching method, which gives excellent results for place recognition.

The passive system achieves its purpose of accurate mapping without active exploration in several ways. First, the mapping process is formulated as one of adaptive estimation of the robot's location. A proper account of estimation ambiguity and update caching allows a least-commitment approach to map adaptation. More importantly, the system allows mapping errors to occur, but diagnoses and corrects these errors when sufficient information becomes available. One of the ways in which this is done is within the estimation process, by viewing estimation as a sort of explanation of observations. This generalizes the usual notion of estimation to allow for explanations which posit previous map errors (and then correct them). A context-sensitive priority mechanism is used to arbitrate between different possible explanations of observations. The other error-correction mechanism in the system is restructuring, which uses long-term statistics on metric observations to discover structural errors in the map and correct them.

The mapper represents the map as a diktiometry, which combines both path and metric information in a single representation. The basis of the representation is the path structure of the environment, rather than the detailed metric layout of walls and objects. This structure is represented as a graph of waypoints, each a locally distinctive location in the world. This representation scheme was chosen because of its robustness with respect to small environmental changes (as noted previously by Kuipers [1978]). Topology does not provide sufficient constraints for passive mapping, so geometric information is added, in the form of set-based constraints between waypoints. Metric information is organized into a reference graph, giving a uniform framework for representing uncertain geometric relationships. In addition to the coarse structure represented by the path graph, finer structure is attached to links between waypoints in the form of link plans. These plans abstract the behavior of the robot in such a way as to allow it to robustly repeat its behavior, even in the face of environmental variation. One of the key ideas is to to use plan-based abstraction for storage of episodic memory, which can be recognized or replayed later.

Passive mapping is not the end of the story, however, since a whole spectrum exists between fully passive and fully active mapping systems. We began the investigation of this spectrum via the use of exploration scripts. Exploration scripts are simple predefined exploration strategies designed to improve the mapper's performance by directing the robot. When a script is applicable, it is suggested to the deliberator which can then decide what to do with it. Exploration, rather than being entirely the prerogative of the mapping system, should be under the control of the deliberator, which can decide whether or not to explore based on the robot's current tasks.

9.1 Conclusions

In summary, this dissertation has presented several specific contributions to the state of the art in robot map learning and place recognition. In brief, we have demonstrated:

- A passive mapping system which can learn maps during task execution by incremental error correction;
- An accurate technique for visual place recognition using image signatures;
- A method for abstracting low-level traces of robot behavior into robust control programs for repeating that behavior;
- Experimentation scripts for intermittent exploration;

- An abstract robot simulator, which is currently being used by AI researchers worldwide for research in learning and planning.

It should be kept in mind that though the research is 'systemic', in that its result is a large integrated mapping system, most of the ideas and techniques may be profitably applied in other contexts. The error correction techniques can easily be combined with other (active) topological mapping techniques (such as those of [Kuipers and Byun, 1988] or [Mataric, 1990]) for increasing robustness. Similarly, the behavior abstraction method is likely to be useful for learning in a variety of robotic applications requiring flexible planned behavior.

9.2 Future Work

Robotic validation

The majority of the work described in this dissertation was developed and tested in simulation. As discussed in Chapter 2, there are three phases of development using simulation: justification, testing, and validation. The Ars Magna simulator is justified, but it has not yet been validated against a robotic implementation of the map-learning system. This validation includes building the infrastructure required to implement the abstract robot model, primarily the construction of the necessary effector modules and interfacing them with the RPL planning language. Effector modules can be viewed as 'competency groups' of lower-level behavioral modules, for example the 'Go to designator' module can be thought of as the composition of 'Orient to designator', 'Go forward', and 'Avoid obstacles' behaviors. The other major piece of development that is required is building waypoint perception routines, both long-range (designation) and short-range (recognition). Our abstract robot model provides useful organizing principles for the development of such low-level robotic capabilities, affording the design of a straightforward research program. Once these capabilities are in place, we can conduct experimentation to determine the correspondence between map-learning in Ars Magna and in the real world. Based on these results, we will be able to both refine the abstract robot model and the mapping system to more closely reflect real-world conditions.

Formalization

Another area for future research is to attempt to formalize more precisely the ideas embodied in the passive map-learning system[1]. This would be attempted within the framework of finite state machine (FSM) inference (the problem being to infer the smallest FSM consistent with a set of observations). One of the fundamental intuitions behind my system is that the geometric structure of the world provides sufficient constraints to make passive mapping tractable (which purely topological mapping is not). A first step towards formalizing the problem starts from noting that geometry may be thought of as giving each state in a finite state machine a unique output. This, by itself, would make the problem trivial. However, these outputs are corrupted by noise. That is, there is a probability that the output the robot sees is not be the output for the current state.

The passive mapping problem may then be formalized as that of inferring an FSM with noisy unique outputs under an unknown, but fixed, stochastic action policy. The main difficulty is in formalizing the notion of goodness of an FSM model. The learner's accuracy with respect to the real FSM depends crucially on the underlying action policy. Simple prediction accuracy does not suffice, because the real FSM may have different regions such that transitions between the regions are rare. In such a case, prediction accuracy may be high as long as the robot stays in a well-explored area; when a transition is made to a region that is not well-explored, prediction accuracy will drop precipitously. This is clearly not desirable—an evaluation metric for the robot's map should not depend on the robot's location. What is required is a notion of prediction accuracy weighted by how well-explored different areas are. Given such a notion, we will be able to develop analysis tools to describe the performance of algorithms for passive learning, and to develop formal algorithms for passive learning.

Behavioral memory

In Chapter 4, we examined the problem of abstracting behavior descriptions for repetition and recognition. There are a number of directions in which this work can be taken, not all related to map-learning. Most obvious is the implementation and

[1]These ideas were developed in conjunction with Dana Angluin.

testing of the task-net abstraction method described in Section 4.5.7. The key idea is that the plan which generates a behavior should be used as a resource in constructing an abstraction of that behavior to store for later recall. Besides its obvious use for case-based planning, this technique might also be applied to knowledge compilation for plan recognition. Plan knowledge gets reorganized by task-net abstraction into descriptions of commonly-applied clichés. If different agents in the world tend to act similarly, then these organized associations can be used as an indexing mechanism for plan recognition.

Intermittent exploration

While we introduced the activity–passivity spectrum here and started to explore it through the use of opportunity scripts, much work remains to be done. More exploration scripts need to be devised, and the ways in which they interact with environmental structure should be analyzed. What sort of scripts work better in what sort of environments? How might good scripts be generated or learned automatically? A more fundamental question is that of the proper tradeoff between exploration and action. As discussed in Chapter 7, exploration has effects both within and without the context of the robot's current task. We need to approach this question from the standpoint of amortized utility, based on the robot's assigned tasks.

We propose to begin studying these issues using the following exploration problem[2]. We model the world as an unknown directed graph, whose nodes are uniquely labeled and whose links have costs associated with traversing them. The robot is given tasks one at a time, where each task consists of a node to visit; each task has some known utility to the robot. When the robot begins, it knows a probability distribution on the possible graphs that it is exploring, as well as a distribution on what tasks it will be assigned. It may also be given a spanning tree for the world (so that it can perform every task, though suboptimally). Our objective is to develop an algorithm which will maximize the robot's total expected utility (utility of tasks minus traversal costs) over a sequence of n tasks. At each step, the robot has the choice to either take the known best step towards its current goal, or to choose an unexplored arc and possibly learn a new short-cut which will improve its efficiency in the future. This choice comes down to comparing the future utility of the two choices. Clearly, as n grows large, it will

[2]Formulated in conjunction with Greg Hager.

eventually be advantageous to simply explore the entire graph at the start; however, it is unclear where this tradeoff occurs. This problem is closely related to the problem of state-space exploration in reinforcement learning [Sutton, 1984]; in that literature random perturbations are used to ensure that the agent eventually explores its space well. Also, there has been some work on similar problems within the paradigm of *competitive learning*, for example, see [Blum and Chalasani, 1993]. Apparently, no one has yet addressed such incremental learning problems with changing tasks or considered expected (rather than worst-case) behavior.

Robust set-based estimation

Although there exist general, well-understood techniques for robust statistical estimation [Hampel *et al.*, 1986], there are no comparable methods for set-based estimation. Since set-based estimation has proven useful for robotics applications [Atiya and Hager, 1991,Hager *et al.*, 1993], developing general robust set-based estimation techniques is of great importance to robotics. One way of viewing a passive mapping system is as a two-fold estimator—it attempts to estimate both the state of the robot and the state of the world at the same time. We may then consider the matcher in our system as a sort of robust estimator, in which the possibility of observations being outliers is explicitly considered. Variations of these techniques may be applicable to general set-based estimation. In the general estimation case, however, we cannot take advantage of the structure of the world which allows the development of the matching priority scheme described in Chapter 5.

The essential idea behind robust estimation of all sorts is that outlying observations should not affect the estimate. One way in which this is done in statistical estimation is to test incoming observations to see if they fit the model implied by the current estimate (assumed correct). If this test fails, observations are rejected as outliers. This idea can easily be applied to set-based estimation by only using observations whose uncertainty sets intersect with the current estimate by more than a threshold. This will reduce the probability of outliers being incorporated into the estimate, and hence improve estimation. The problem is that if the estimated set no longer contains the true state (due to outliers), it will never contain the true state. Hence, an analog of **E-Match** or **N-Match** (refer to Chapter 5) is required to 'reset' the estimate sometimes. This might be done by expanding the estimate set slightly when a non-intersecting, but nearby observation is made. Thus, if the estimate is corrupted by an outlier, it will tend back towards true under the influence of subsequent good observations. These ideas need to be explored more formally, both by analysis of simple cases and simulation.

9.3 A Modest Proposal

Where does the future of map learning research lie? In addition to the passive mapper described in this dissertation, in Chapter 8 we discussed a number of map-learning systems, each with its own input-output specification and assumptions about the world. All of these systems have been shown to work well in certain cases. However, it is extremely difficult to evaluate different systems and compare them to one another. Systems are implemented on widely varying robotic platforms, experiments are performed in very different types of environments, and system components are interdependent so it is hard to understand how they individually affect performance. All of these are problems for many areas of robotic research besides map learning, but we will focus on map learning here.

Evaluation

There are serious methodological problems with the way robotic research is currently evaluated. Typically, several test problems are set up using a particular robotic platform, and the system's performance on those problems is recorded. This makes it very difficult to tell how the system will perform if it is implemented on a different platform or run in a different environment. A key problem facing robotics today therefore is methodological—we need effective tools for empirical evaluation and analysis of proposed techniques and systems. In Chapter 2, we suggested the judicious use of properly designed simulation environments for AI research. We believe that they can also be useful in addressing this problem of generality in robotic evaluation. Properly designed simulated environments can give the researcher control over many aspects of their experiments, permitting more detailed analyses than current techniques permit. Such simulations would allow flexible configuration of robotic platforms, including variation of sensor/effector noise models, as well as variability in the structure and materials composing the robot's environment. This control, if used systematically, will permit more general definite statements about the operation of a mapping system than, "We tried it in our building and it seemed to work."

As discussed in Chapter 2, however, simulation must be used with care, if results are to be relevant to the real world. The three-phase experimental methodology of justification, testing, and validation must be followed. Simulation design must be justified by current robotic practice. Proposed mapping systems (or specific techniques) should be tested extensively in the simulation system to evaluate their effectiveness. And finally, simulated results should be validated against comparable real-world results in order to verify the relevance of the simulation. This is the proper methodology to use now in studying robotic map learning, as well as in robotics as a whole. It is imperative to develop a general simulation testbed environment that incorporating sufficient variability to support the experimentation life-cycle we described.

Analysis

Since many useful map learning systems have already been described in the literature, we propose to carry out a large-scale comparative study of these extant methods, using validated simulation as above. There are two main questions that this research will address. Most fundamental is the question of what assumptions different methods make about the sensors, effectors, and the environment. This requires careful evaluation and analysis, first of all because not all assumptions are explicit or obvious and must be discovered through experimentation. Secondly, even though a mapping technique may formally depend on some strong assumptions, in practice those assumptions may not be very important. For example, Elfes' [1987] application of certainty grids to sonar-based mapping makes the unrealistic assumption that sonar reflections are Lambertian; however, the system works well locally because of the density of the the representation and of observations. By carefully examining the behavior of different mapping systems in different environments with different sensors, we will be able to better understand both what assumptions the systems make and how those assumptions affect their results. Since, at present, it is rare for researchers in robotics to work at replicating others' results, this kind of large-scale comparative study is long overdue.

In addition to analyzing the 'black box' behavior of mapping systems, treating them as complete independent systems, we can also work at understanding the relationships between different systems and among components of single systems. For example, Leonard *et al.*'s [1992] sonar-based mapping system uses a Bayesian method for estimation and outlier rejection; we might compare that with a set-based estimator in the same framework. This would allow us to better understand where the power of the system comes from. In addition, as we learn to modularize such systems, we can better understand what specific techniques are more appropriate in what situations, and how to combine techniques to best effect.

Synthesis

The understanding gained from the analytical phase of the project will lead directly to several important technological results. The first technological goal is the development of an engineering methodology for building robotic learning systems. Understanding the properties and relationships of different mapping techniques will enable the roboticist to choose the right ensemble of methods for a specific target platform and environment. After we have gained some experience in building new learning systems, we may then design software tools to support such development. Such tools would include system architectures configurable for different sensor configurations and environment models, as well as implementations of the different models themselves. For example, a generic 'geometric features' mapping system could be configured to work with sonar or with stereo line data (or both). Finally, we would like to work on developing methods for figuring out the relevant characteristics of the environment, so as to choose appropriate representation and sensing strategies automatically. A combination of careful experimentation (with judicious use of simulation) and engineering development will contribute greatly to the development of fully autonomous robots.

Bibliography

[Agre and Chapman, 1987] P.E. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of AAAI-87, Seattle, Wa.*, pages 268–272. AAAI, July 1987.

[Agre and Chapman, 1988] Philip E. Agre and David Chapman. What are plans for? Technical Report A.I. Memo 1050, MIT AI Lab, September 1988.

[Agre, 1988] Philip E. Agre. *The Dynamic Structure of Everyday Life.* PhD thesis, MIT Artificial Intelligence Laboratory, 1988.

[Al-Badr and Hanks, 1991] Badr Al-Badr and Steve Hanks. Critiquing the Tileworld: Agent architectures, planning benchmarks, and experimental methodology. *AI Magazine*, 1991. submitted.

[Alefeld and Hertzberger, 1983] G. Alefeld and J. Hertzberger. *Introduction to Interval Computation.* Academic Press, New York, NY, 1983.

[Angluin, 1978] Dana Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, (39), 1978.

[Angluin, 1987] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, November 1987.

[Arkin, 1987] Ronald C. Arkin. *Towards Cosmopolitan Robots: Intelligent Navigation in Extended Man-Made Environments.* PhD thesis, University of Massachusetts, Department of Computer and Information Sciences, 1987. COINS Technical Report 87-80.

[Arkin, 1989] Ronald C. Arkin. Motor schema-based mobile robot navigation. *International Journal of Robotics Research*, 1989.

[Atiya and Hager, 1991] Sami Atiya and Greg Hager. Real-time vision-based robot localization. In *Proc. Int'l Conf. on Robotics and Automation*, 1991.

[Ballard and Brown, 1982] Dana H. Ballard and Christopher M. Brown. *Computer Vision*. Prentice-Hall, 1982.

[Ballard and Ozcandarli, 1988] Dana H. Ballard and Altan Ozcandarli. Eye fixation and early vision: Kinetic depth. In *Proc. Int'l Conf. on Computer Vision*, 1988.

[Ballard, 1989] Dana H. Ballard. Reference frames for animate vision. In *Proc. Int'l Joint Conference on Artificial Intelligence*, 1989.

[Barmish and Sankaran, 1979] B. Ross Barmish and Jaishankar Sankaran. The propagation of parametric uncertainty via polytopes. *IEEE Transactions on Automatic Control*, AC-24(2), 1979.

[Basye *et al.*, 1989] Kenneth Basye, Tom Dean, and Jeffrey S. Vitter. Coping with uncertainty in map learning. In *Proc. Int'l Joint Conference on Artificial Intelligence*, 1989.

[Berger, 1985] James O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer-Verlag, New York, 1985.

[Bertsekas and Rhodes, 1971] Dimitri P. Bertsekas and Ian B. Rhodes. Recursive state estimation for a set-membership description of uncertainty. *IEEE Transactions on Automatic Control*, AC-16(2), 1971.

[Black, 1992] Michael J. Black. *Robust Incremental Optical Flow*. PhD thesis, Yale University, September 1992. Technical Report 923.

[Blum and Chalasani, 1993] Avrim Blum and Prasad Chalasani. An on-line algorithm for improving performance in navigation. In *Proc. Symp. Foundations of Computer Science*, 1993.

[Brand *et al.*, 1993] Matthew Brand, Lawrence Birnbaum, and Paul Cooper. Sensible scenes: Visual understanding of complex structures through causal analysis. In *Proc. National Conference on Artificial Intelligence*, 1993.

[Braunegg, 1990] David J. Braunegg. *MARVEL: A System for Recognizing World Locations with Stereo Vision*. PhD thesis, MIT, 1990.

[Brogan, 1985] William L. Brogan. *Modern Control Theory*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1985.

[Brooks, 1985] Rodney A. Brooks. Visual map making for a mobile robot. In *Proc. Int'l Conf. on Robotics and Automation*, 1985.

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

[Byun, 1990] Yung-Tai Byun. *Spatial Learning Mobile Robots With A Spatial Semantic Hierarchical Model*. PhD thesis, University of Texas, Austin, 1990. Artificial Intelligence Lab Technical Report AI90-121.

[Chapman, 1987] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 1987.

[Chatila and Laumond, 1985] Raja Chatila and J. Laumond. Position referencing and consistent world modeling for mobile robots. In *Proc. Int'l Conf. on Robotics and Automation*, pages 138–170, Washington, D.C., 1985.

[Chernoff, 1979] John Miller Chernoff. *African Rhythm and African Sensibility*. The University of Chicago Press, Chicago, IL, 1979.

[Chrisman, 1992] Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proc. National Conference on Artificial Intelligence*, pages 183–188, 1992.

[Cohen *et al.*, 1989] Paul R. Cohen, Michael L. Greenberg, David M. Hart, and Adele E. Howe. Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine*, 10(3), 1989.

[Connell, 1989] Jonathan Connell. *A Colony Architecture for an Artificial Creature*. PhD thesis, MIT, 1989. Technical Report 1151.

[Cormen *et al.*, 1992] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1992.

[Crowley, 1984] James Crowley. Dynamic world modeling for an intelligent mobile robot. In *Proceedings of ICPR 7*, pages 207–210, 1984.

[Crowley, 1985] James L. Crowley. Navigation for an intelligent mobile robot. *IEEE Journal of Robotics and Automation*, RA-1(1), 1985.

[Davis, 1984] Ernest Davis. *Representing and Acquiring Geographic Knowledge*. PhD thesis, Yale University Department of Computer Science, 1984.

[Dean *et al.*, 1992] Tom Dean, Dana Angluin, Kenneth Basye, Sean Engelson, Leslie Kaelbling, Evangelos Kokkevis, and Oded Maron. Inferring finite automata with stochastic output functions and an application to map learning. In *Proc. National Conference on Artificial Intelligence*, pages 208–214, 1992.

[Dudek *et al.*, 1988] Gregory Dudek, Michael Jenkins, Evangelos Milios, and David Wilkes. Robotic exploration as graph construction. Technical Report RBCV-TR-88-23, University of Toronto, 1988.

[Elfes, 1987] Alberto Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, RA-3(3):249–265, 1987.

[Engelson and Bertani, 1992] Sean P. Engelson and Niklas Bertani. ARS MAGNA: The abstract robot simulator manual. Technical Report YALEU/DCS/TR-928, Yale University Department of Computer Science, 1992.

[Engelson and McDermott, 1991] Sean P. Engelson and Drew V. McDermott. Image signatures for place recognition and map construction. In *Proceedings of SPIE Symposium on Intelligent Robotic Systems, Sensor Fusion IV*, 1991.

[Engelson, 1990] Sean P. Engelson. Perceptual integration of piloting and navigation. In *Proceedings of IEEE Int'l Symp. on Intelligent Control*, 1990.

[Engelson, 1992] Sean P. Engelson. Active place recognition using image signatures. In *Proceedings of SPIE Symposium on Intelligent Robotic Systems, Sensor Fusion V*, 1992.

[Erdman and Mason, 1986] Michael A. Erdman and Matthew T. Mason. An exploration of sensorless manipulation. In *Proc. Int'l Conf. on Robotics and Automation*, 1986.

[Fikes and Nilsson, 1971] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Firby and Hanks, 1987] R. James Firby and Steve Hanks. The simulator manual. Technical Report YALEU/DCS/TR-563, Yale University Department of Computer Science, 1987.

[Firby, 1989] R. James Firby. *Adaptive Execution in Complex Dynamic Worlds*. PhD thesis, Yale University, January 1989. Technical Report 672.

[Fryer, 1966] H. C. Fryer. *Concepts and Methods of Experimental Statistics*. Allyn and Bacon, Inc., Boston, 1966.

[Gallistel, 1990] Charles R. Gallistel. *The Organization of Learning*. The MIT Press, Cambridge, MA, 1990.

[Gat, 1991] Erann Gat. *Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots.* PhD thesis, Virginia Polytechnic Institute and State University, 1991.

[Gat, 1993] Erann Gat. On the role of stored internal state in the control of autonomous mobile robots. *AI Magazine*, 14(1), 1993.

[Gelb, 1974] Arthur Gelb, editor. *Applied Optimal Estimation.* The MIT Press, Cambridge, MA, 1974.

[Gold, 1978] E. Mark Gold. Complexity of automaton identification from given sets. *Information and Control*, (37), 1978.

[Hager *et al.*, 1993] Gregory Hager, Sean Engelson, and Sami Atiya. On comparing statistical and set-based methods in sensor data fusion. In *Proc. Int'l Conf. on Robotics and Automation*, Atlanta, GA, May 1993.

[Hager *et al.*, 1994] Gregory D. Hager, Wen-Chung Chang, and Steven Morse. Robot feedback control based on stereo vision: Towards calibration-free hand-eye coordination. In *Proc. Int'l Conf. on Robotics and Automation*, 1994. (submitted).

[Hager, 1990] Gregory D. Hager. *Task-Directed Sensor Fusion and Planning.* Kluwer, Boston, MA, 1990.

[Hampel *et al.*, 1986] F. R. Hampel, E. M. Ronchetti, P. J. Rousseeuw, and W. A. Stahel. *Robust Statistics: The Approach Based on Influence Functions.* John Wiley & Sons, New York, NY, 1986.

[Haralick *et al.*, 1973] Robert M. Haralick, K. Shanmugam, and Its'hak Dinstein. Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics*, 3(6), 1973.

[Hong *et al.*, 1991] Jiawei Hong, Xiaonan Tan, Brian Pinette, Richard Weiss, and Edward M. Riseman. Image-based homing. In *Proc. Int'l Conf. on Robotics and Automation*, 1991.

[Horswill, 1993] Ian Horswill. *Specialization of Perceptual Processes.* PhD thesis, MIT Artificial Intelligence Laboratory, 1993.

[Khatib, 1985] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proc. Int'l Conf. on Robotics and Automation*, pages 500–505, 1985.

[Kirman *et al.*, 1991] Jak Kirman, Kenneth Basye, and Thomas Dean. Sensor abstraction for control of navigation. In *Proc. Int'l Conf. on Robotics and Automation*, Sacramento, CA, 1991.

[Koditschek, 1989] Daniel E. Koditschek. Autonomous mobile robots controlled by navigation functions. In *The Second IEEE International Workshop on Intelligent Robots and Systems*, September 1989.

[Koenderink and van Doorn, 1989] Jan J. Koenderink and Andrea J. van Doorn. Photometric invariants related to solid shape. In Berthold K. P. Horn and Michael J. Brooks, editors, *Shape From Shading*, pages 301–322. MIT Press, 1989.

[Kortenkamp *et al.*, 1992] David Kortenkamp, L. Douglas Baker, and Terry Weymouth. Using gateways to build a route map. In *Proc. IEEE/RSJ Int'l Workshop on Intelligent Robots and Systems*, 1992.

[Kriegman, 1989] David J. Kriegman. *Object Classes and Image Contours in Model-Based Vision.* PhD thesis, Stanford University, 1989.

[Kuc and Viard, 1991] Roman Kuc and Victor Brian Viard. A physically based navigation strategy for sonar-guided vehicles. *The International Journal of Robotics Research*, 10(2), 1991.

[Kuipers and Byun, 1988] Benjamin Kuipers and Yung-Tai Byun. A robust qualitative method for robot spatial reasoning. In *Proc. National Conference on Artificial Intelligence*, pages 774–779, 1988.

[Kuipers, 1978] Benjamin Kuipers. Modeling spatial knowledge. *Cognitive Science*, 2:129–153, 1978.

[Leonard *et al.*, 1992] 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):286–298, 1992.

[Levitt *et al.*, 1987] T. S. Levitt, D. T. Lawton, D. M. Chelberg, and P. C. Nelson. Qualitative landmark-based path planning and following. In *Proc. National Conference on Artificial Intelligence*, Seattle, Washington, 1987.

[Lumelsky and Stepanov, 1987] V. J. Lumelsky and A. A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, (2):403–430, 1987.

[Lynch, 1960] Kevin Lynch. *The Image of the City.* The MIT Press, Cambridge, MA, 1960.

[Maes, 1989] Pattie Maes. How to do the right thing. *Connection Science*, 1, 1989.

[Malkin and Addanki, 1992] Malkin and Addanki. Lognets. In *Proc. National Conference on Artificial Intelligence*, 1992.

[Mataric, 1990] Maja J. Mataric. A distributed model for mobile robot environment-learning and navigation. Technical Report 1228, MIT Artificial Intelligence Laboratory, 1990.

[McAllester and Rosenblitt, 1991] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proc. National Conference on Artificial Intelligence*, pages 634–639, 1991.

[McDermott and Davis, 1984] Drew V. McDermott and Ernest Davis. Planning routes through uncertain territory. *Artificial Intelligence*, 22:107–156, 1984.

[McDermott, 1978] Drew V. McDermott. Planning and acting. *Cognitive Science*, (2):71–109, 1978.

[McDermott, 1981a] Drew McDermott. Artificial intelligence meets natural stupidity. In *Mind Design*, pages 143–160. The MIT Press, 1981.

[McDermott, 1981b] Drew McDermott. Finding objects with given spatial properties. Technical Report 195, Yale University Department of Computer Science, 1981.

[McDermott, 1988] Drew McDermott. Revised NISP manual. Technical Report 642, Yale University Department of Computer Science, 1988.

[McDermott, 1990] Drew V. McDermott. Planning reactive behavior: A progress report. In *Proc. DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, 1990.

[McDermott, 1991] Drew McDermott. A reactive plan language. Technical Report 864, Yale University Department of Computer Science, 1991.

[McDermott, 1992] Drew V. McDermott. Transformational planning of reactive behavior. Technical Report YALEU/CSD/RR #941, Yale University Department of Computer Science, 1992.

[McLachlan and Basford, 1988] Geoffrey J. McLachlan and Kaye E. Basford. *Mixture Models: Inference And Applications To Clustering.* M. Dekker, New York, 1988.

[Mitchell *et al.*, 1986] T.M. Mitchell, R.M. Keller, and S.T. Kedar-Cabelli. Explanation–based generalization: A unifying view. *Machine Learning*, 1(1):47–80, 1986.

[Moravec and Elfes, 1985] Hans P. Moravec and Alberto Elfes. High resolution maps from wide angle sonar. In *Proc. Int'l Conf. on Robotics and Automation*, pages 138–145, 1985.

[Nelson and Aloimonos, 1988] Randall Nelson and John Aloimonos. Obstacle avoidance: Towards qualitative vision. In *Proc. Int'l Conf. on Computer Vision*, 1988.

[Nelson, 1989] Randall Nelson. *Visual Navigation.* PhD thesis, Computer Vision Laboratory, University of Maryland, 1989.

[Neumaier, 1990] Arnold Neumaier. *Interval Methods for Systems of Equations.* Cambridge University Press, 1990.

[Norman, 1980] Donald A. Norman. *The Psychology of Everyday Things.* Basic Books, New York, NY, 1980.

[Payton, 1986] David W. Payton. An architecture for reflexive autonomous vehicle control. In *Proc. Int'l Conf. on Robotics and Automation*, pages 1838–1845, 1986.

[Pearl, 1988] Judea Pearl. *Probabilistic Reasoning In Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufman, San Mateo, CA, 1988.

[Pentland, 1989] Alex P. Pentland. Local shading analysis. In Berthold K. P. Horn and Michael J. Brooks, editors, *Shape From Shading*, pages 443–488. MIT Press, 1989.

[Philips and Bresina, 1991] A. Philips and J. Bresina. NASA Tileworld manual. Technical Report TR-FIA-91-11, NASA Ames Research Center, Code FIA, 1991.

[Pitt and Warmuth, 1989] Leonard Pitt and Manfred K. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. In *Proc. Twenty First Annual ACM Symp. on Theoretical Computing*, 1989.

[Pollack and Ringuette, 1990] Martha E. Pollack and Marc Ringuette. Introducing the Tileworld: Experimentally evaluating agent architectures. In *Proc. National Conference on Artificial Intelligence*, 1990.

[Preparata and Shamos, 1988] Franco Preparata and Michael Ian Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 2nd edition, 1988.

[Ramey, 1982] Daniel Bruce Ramey. *A Non-Parametric Test Of Bimodality With Applications To Cluster Analysis.* PhD thesis, Yale University Department of Statistics, 1982.

[Rivest and Schapire, 1989] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. In *Proc. 21st Ann. ACM Symp. on Theoretical Computing*, 1989.

[Rivest and Sloan, 1988] Ronald L. Rivest and Robert Sloan. Learning complicated concepts reliably and usefully. In *Proc. National Conference on Artificial Intelligence*, pages 635–640, 1988.

[Rosenblatt and Payton, 1989] J. Kenneth Rosenblatt and David Payton. A fine-grained alternative to the subsumption architecture for mobile robot control. In *Proc. Int'l Joint Conference on Neural Networks*, 1989.

[Sarachik, 1989] Karen B. Sarachik. Visual navigation: Constructing and utilizing simple maps of an indoor environment. Technical Report 1113, MIT Artificial Intelligence Laboratory, 1989.

[Schank and Abelson, 1977] Roger C. Schank and Robert Abelson. *Scripts, Plans, Goals and Understanding.* Erlbaum, Hillsdale, NJ, 1977.

[Schank, 1982] Roger C. Schank. *Dynamic Memory.* Cambridge University Press, 1982.

[Schweppe, 1968] Fred C. Schweppe. Recursive state estimation: Unknown but bounded errors and system inputs. *IEEE Transactions on Automatic Control*, AC-13(5), 1968.

[Slack, 1990] Marc Slack. *Situationally Driven Local Navigation for Mobile Robots.* PhD thesis, Virginia Polytechnic Institute, 1990. NASA JPL Publication 90-17.

[Smith *et al.*, 1986] Randall Smith, Matthew Self, and Peter Cheeseman. Estimating uncertain spatial relationships in robotics. In *Proceedings of the Second Workshop on Uncertainty in Artificial Intelligence*, Philadelphia, PA, 1986.

[Sutton, 1984] R. Sutton. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts at Amherst, 1984.

[Swain, 1990] Michael J. Swain. *Color Indexing.* PhD thesis, University of Rochester, 1990.

[Tarabanis *et al.*, 1990] Konstantinos Tarabanis, Roger Y. Tsai, and Peter K. Allen. Automated sensor planning and modeling for robotic vision tasks. Technical Report CUCS 045-90, Columbia University, New York, NY, 1990.

[Taylor and Kriegman, 1993] Camillo J. Taylor and David J. Kriegman. Exploration strategies for mobile robots. In *Proc. Int'l Conf. on Robotics and Automation*, Atlanta, GA, May 1993.

[Torrance, 1994] Mark C. Torrance. Natural communication with robots. Master's thesis, MIT Artificial Intelligence Laboratory, 1994.

[Vere and Bickmore, 1990] Steven Vere and Timothy Bickmore. A basic agent. *Computational Intelligence*, 6(1), Feb 1990.

[Watkins, 1989] C. J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, University of Cambridge, 1989.

[Witkin, 1981] Andrew P. Witkin. Recovering surface shape and orientation from texture. *Artificial Intelligence*, 17:17–47, 1981.

[Witsenhausen, 1968a] H. S. Witsenhausen. A minimax control problem for sampled linear systems. *IEEE Transactions on Automatic Control*, AC-13(1), 1968.

[Witsenhausen, 1968b] H. S. Witsenhausen. Sets of possible states of linear systems given perturbed observations. *IEEE Transactions on Automatic Control*, AC-13(5), 1968.

[Yamauchi and Nelson, 1991] Brian Yamauchi and Randall Nelson. A behavior-based architecture for robots using real-time vision. In *Proc. Int'l Conf. on Robotics and Automation*, 1991.

[Yeap, 1988] Wai K. Yeap. Towards a computational theory of cognitive maps. *Artificial Intelligence*, 34(3), April 1988.

[Yemini, 1979] Yechiam Yemini. Some theoretical aspects of position-location problems. In *Proc. 20th Symp. on Foundations of Computer Science*, pages 1–8, 1979.

[Zetsche and Caelli, 1989] Christoph Zetsche and Terry Caelli. Invariant pattern recognition using multiple filter image representations. *Computer Vision, Graphics, and Image Processing*, 45:251–262, 1989.

Appendix A

Interval Arithmetic

In this appendix, I will describe the basic concepts of interval arithmetic that are used in the mapping system described in this dissertation. The interested reader is referred to [Neumaier, 1990] and [Alefeld and Hertzberger, 1983] for more information on interval mathematics.

A.1 Interval Basics

A *real interval* is a set of real numbers of the form

$$\mathbf{x} = [\underline{x}, \overline{x}] = \{a \mid \underline{x} \leq a \leq \overline{x}\}$$

The set of all real intervals is denoted by $\mathbb{IR}$. We may also talk about the *midpoint* of an interval $\mathbf{x}$, $\mathrm{mid}(\mathbf{x}) = (\overline{x} + \underline{x})/2$, and its *radius*, $\mathrm{rad}(\mathbf{x}) = (\overline{x} - \underline{x})/2$. If S is any nonempty, bounded subset of $\mathbb{R}$, we call the (unique) minimal interval containing S the *hull* of S, denoted $\square S$. The hull is also often called the *least bounding interval*, or LBI.

We can extend the order relations $\propto \in \{<, \leq, >, \geq\}$ to interval arguments by universal quantification:

$$\mathbf{x} \propto \mathbf{y} \equiv a \propto b \quad \forall a \in \mathbf{x}, b \in \mathbf{y}$$

Note that while the order relations remain antisymmetric and transitive on intervals, they are not complete, for not all pairs of intervals are comparable, for example [0,2] and [1,3].

While the intersection of any two intervals $\mathbf{x}$ and $\mathbf{y}$, $\mathbf{x} \cap \mathbf{y}$, is an interval (possibly empty), their union may not be. We thus define the *hull union* of $\mathbf{x}$ and $\mathbf{y}$ as $\mathbf{x} \sqcup \mathbf{y} = \square(\mathbf{x} \cup \mathbf{y})$.

A.1.1 Interval arithmetic

We can extend arithmetic operations to operate on intervals, by defining, for intervals $\mathbf{x}$ and $\mathbf{y}$, and arithmetic operators $\circ \in \{+, -, *, /\}$:

$$\mathbf{x} \circ \mathbf{y} = \square\{a \circ b \mid a \in \mathbf{x}, b \in \mathbf{y}\} = \{a \circ b \mid a \in \mathbf{x}, b \in \mathbf{y}\}$$

provided that $\circ$ is defined for all elements of $\mathbf{x}$ and $\mathbf{y}$. So, for example, division is undefined for intervals containing zero.

By using continuity and monotonicity, we can express the results of operations on intervals in terms of their endpoints, as for addition and subtraction:

$$\begin{aligned} \mathbf{x} + \mathbf{y} &= [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \\ \mathbf{x} - \mathbf{y} &= [\underline{x} - \overline{y}, \overline{x} - \underline{y}] \end{aligned}$$

Multiplication and division are a bit more complicated, as the results depend on the signs of the intervals:

$$\mathbf{xy} \quad = \quad \begin{array}{c|lll} & \mathbf{y} \geq 0 & 0 \in \mathbf{y} & \mathbf{y} \leq 0 \\ \hline \mathbf{x} \geq 0 & [\underline{x}\underline{y}, \overline{x}\overline{y}] & [\overline{x}\underline{y}, \overline{x}\overline{y}] & [\overline{x}\underline{y}, \underline{x}\overline{y}] \\ 0 \in \mathbf{x} & [\underline{x}\overline{y}, \overline{x}\overline{y}] & \square\{\overline{x}\underline{y}, \underline{x}\overline{y}\} & [\overline{x}\underline{y}, \underline{x}\underline{y}] \\ \mathbf{x} \leq 0 & [\underline{x}\overline{y}, \overline{x}\underline{y}] & [\underline{x}\overline{y}, \underline{x}\underline{y}] & [\overline{x}\overline{y}, \underline{x}\underline{y}] \end{array}$$

$$\mathbf{x}/\mathbf{y} \quad = \quad \begin{array}{c|ll} & \mathbf{y} > 0 & \mathbf{y} < 0 \\ \hline \mathbf{x} \geq 0 & [\underline{x}/\overline{y}, \overline{x}/\underline{y}] & [\overline{x}/\overline{y}, \underline{x}/\underline{y}] \\ 0 \in \mathbf{x} & [\underline{x}/\underline{y}, \overline{x}/\underline{y}] & [\overline{x}/\overline{y}, \underline{x}/\overline{y}] \\ \mathbf{x} \leq 0 & [\underline{x}/\underline{y}, \overline{x}/\overline{y}] & [\overline{x}/\underline{y}, \underline{x}/\overline{y}] \end{array}$$

A.1.2 Modulo intervals

We can also consider the space of intervals over $\mathbb{R}_k$, the reals modulo k; we term this space $\mathbb{IR}_k$, the *modulo k intervals*. For robotics, we are most interested in the *angle intervals* $\mathbb{I}\mathbf{A} = \mathbb{IR}_{2\pi}$ (this is just S^1 by another name). Modulo interval arithmetic is simple; for $\mathbf{x}, \mathbf{y} \in \mathbb{IR}_k$, $\mathbf{x} \circ \mathbf{y} \bmod k = [\underline{z} \bmod k, \overline{z} \bmod k]$, where $\mathbf{z} = \mathbf{x} \circ \mathbf{y}$. (When $\mathbf{x}$ and $\mathbf{y}$ are known to be modulo intervals, we drop the "mod k" on the left-hand side.) If a modulo k interval $[a, b]$ has $a > b$, it straddles k, so $[a, b] = [a, k] \cup [0, b]$. By convention, $[a, a]$ denotes the thin interval only including a, and $[0, k]$ the complete interval on $\mathbb{R}_k$.

The fact that $\mathbb{R}_k$ has a different topology from $\mathbb{R}$ affects our treatment of intervals over $\mathbb{R}_k$. First, the hull of a set of points in $\mathbb{R}_k$ is not uniquely defined. For example, take the set of points $S_1 = \{0, 2, 2.5\} \subset \mathbb{R}_3$. The hull of S_1 in $\mathbb{IR}_3$ may be legitimately taken as $[0, 2.5]$, $[2, 0]$, or $[2.5, 2]$. This ambiguity may be partially resolved by defining $\Box S$ of a subset of $\mathbb{R}_k$ as the *smallest* hull of S—thus, $\Box S_1 = [2, 0]$. This does not entirely resolve the ambiguity; consider for example the set $\{0, 1, 2\} \subset \mathbb{R}_3$. In such cases the ambiguity is inherent; any choice of hull is permissible. Care should be taken, of course, to avoid implicitly assuming that hulls are unique; absent uniquification in an implementation (which is recommended), hulls of equal sets need not be equal.

The second interval operation which is complicated by $\mathbb{R}_k$'s topology is interval intersection. Unlike the intersection of real intervals, the intersection of two modulo intervals is not necessarily another interval (consider the example of $[0, 2.5] \cap [2, 0.5]$ in $\mathbb{IR}_3$). One way of resolving this difficulty is by defining the *hull intersection* of intervals $\mathbf{x}$ and $\mathbf{y}$ by $\mathbf{x} \sqcap \mathbf{y} = \Box(\mathbf{x} \cap \mathbf{y})$. (So, we would have $[0, 2.5] \sqcap [2, 0.5] = [2, 0.5]$.) Unlike hull union, however, hull intersection is not associative, so we explicitly define the hull intersection of a set $\{\mathbf{x}_i\}$ of modulo intervals by

$$\bigsqcap_i \mathbf{x}_i = \Box(\bigcap_i \mathbf{x}_i) \ .$$

The hull intersection of a set of intervals is never larger than the smallest interval in the set, though it may be as large.

An alternative to hull intersecting $\mathbf{x}$ and $\mathbf{y}$ is to use their *directed hull intersection*, $\mathbf{x} \overleftarrow{\sqcap} \mathbf{y}$, which is the smallest interval containing $\mathbf{x} \cap \mathbf{y}$ which is a subset of $\mathbf{x}$. Though the directed hull intersection is at least as large as the hull intersection, it can be useful for robustifying recursive estimation.

A.2 Interval Functions

In the same way as we extended arithmetic operations to intervals in the last section, we can define the *interval extension* of an arbitrary real-valued function f:

$$\mathbf{f}^*(\mathbf{x}) = \Box\{f(a) \mid a \in \mathbf{x}\}$$

As long as f is continuous, $\mathbf{f}^*(\mathbf{x})$ will contain exactly the image of $\mathbf{x}$ under f (so the $\Box$ operator is redundant). In particular, this is the case for the arithmetic operators described above. If this is not the case, $\mathbf{f}^*(\mathbf{x})$ will include values not in $f(\mathbf{x})$ (if, for example, $f(\mathbf{x})$ is a disconnected set); this is called *overestimation*. Overestimation can also stem from computational considerations, as we will see below.

A.2.1 Interval evaluation

Suppose we consider functions that can be expressed as composition of elementary arithmetic operations, such as $f_1(x, y, z) = x(y + 2z)$. One way to approximate $\mathbf{f}_1^*(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \{a(b+2c) \mid a \in \mathbf{x}, b \in \mathbf{y}, c \in \mathbf{z}\}$, is by evaluating f_1's expression, $x(y+2z)$, for intervals $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$, replacing arithmetic operations by their interval equivalents; we denote the interval function that results $\mathbf{f}_1$. This approximation method is called *interval evaluation*; it is clear that for any function $f(x)$, $\mathbf{f}^*(\mathbf{x}) \subset \mathbf{f}(\mathbf{x})$, so $\mathbf{f}$ is consistent with $\mathbf{f}^*$. Interval evaluation is simple and efficient, and works well for f_1, giving us exactly $\mathbf{f}_1(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \mathbf{f}_1^*(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \{a(b + 2c) \mid a \in \mathbf{x}, b \in \mathbf{y}, c \in \mathbf{z}\}$. Other functions are not so simple, for example, take $f_2(x, y) = x - xy$. If we compute $\mathbf{f}_2([1, 2], [3, 5])$ by arithmetic extension as above, we get $[-9, -1]$. However, $\mathbf{f}_2^*([1, 2], [3, 5]) = \{a - ab \mid a \in [1, 2], b \in [3, 5]\} = [-8, -2]$. This illustrates the overestimation inherent in interval estimation whenever an expression contains the same variable more than once. Interval estimation 'forgets' that multiple intervals corresponding to the same variable are correlated, which can result in overestimation.

A.2.2 Mean value forms

Better approximations to $\mathbf{f}^*$ than $\mathbf{f}$ can be achieved, for narrow intervals, by using a method based on the mean value theorem. Let f have computable derivatives. Now, consider an interval $\mathbf{x}$ as an approximation to a point value, $\mathbf{x} = \mathrm{mid}(\mathbf{x}) \pm \mathrm{rad}(\mathbf{x})$. We can then use the mean value theorem to calculate bounds on the value of $f(x')$ for arbitrary x':

$$f(x') = f(\mathrm{mid}(\mathbf{x})) + f'(\xi)(x' - \mathrm{mid}(\mathbf{x}))$$

for some ξ between x' and $\mathrm{mid}(\mathbf{x})$. Hence, since $x' \in \mathbf{x}$ implies that $\xi \in \mathbf{x}$, we can compute the *mean value form* for f

$$\mathbf{f}_m(\mathbf{x}) = f(\mathrm{mid}(\mathbf{x})) + \mathbf{f}'(\mathbf{x})(\mathbf{x} - \mathrm{mid}(\mathbf{x}))$$

Since $\mathbf{f}^*(\mathbf{x}) \subset \mathbf{f}_m(\mathbf{x})$, we can use $\mathbf{f}_m$ as an approximation to $\mathbf{f}^*$.

A.2.3 Extremal analysis

Both interval evaluation and the use of mean value forms for approximating functions' interval extensions introduce overestimation, though they are very general techniques. In specific cases, however, we can usually do much better, and often we can compute $\mathbf{f}^*$ exactly. Extremal analysis is a technique that can be used to derive a procedure for the exact interval extensions of a continuous function with a countable non-dense set of extrema. Let f be such a function, with extrema $E = \{y_i\}$. Suppose we wish to compute $\mathbf{f}^*(\mathbf{x})$, for some interval $\mathbf{x}$. If $\mathbf{x}$ contains no extrema, then it falls over a monotone region of f's domain, and $\mathbf{f}^*(\mathbf{x}) = [\min(f(\underline{x}), f(\overline{x})), \max(f(\underline{x}), f(\overline{x}))]$. Otherwise, let $E_x = E \cap \mathbf{x}$. This gives us

$$\mathbf{f}^*(\mathbf{x}) = [\min(f(\underline{x}), f(\overline{x}), \min_{y \in E_x} f(y)), \max(f(\underline{x}), f(\overline{x}), \max_{y \in E_x} f(y))]$$

since f's extrema over $\mathbf{x}$ must either be at $\mathbf{x}$'s endpoints or in E_x. This method only works, of course, if E_x can be computed. For functions with a finite number of extrema, E can be precomputed, giving a simple conditional procedure can be used to evaluate $\mathbf{f}^*$. For example, for $f(x) = x^2$, we have:

- If $\mathbf{x} > 0$ then $\mathbf{f}^*(\mathbf{x}) = [\underline{x}^2, \overline{x}^2]$;

- If $\mathbf{x} < 0$ then $\mathbf{f}^*(\mathbf{x}) = [\overline{x}^2, \underline{x}^2]$;
- If $0 \in \mathbf{x}$ then $\mathbf{f}^*(\mathbf{x}) = [0, \max(\underline{x}^2, \overline{x}^2)]$.

The observant reader will note that this is exactly the method which was used to arrive at the formulae above describing interval multiplication and division. The complexity of such a procedure is $\mathrm{O}(|E|)$.

A.3 Vectors and Matrices

An n-dimensional real interval vector $\mathbf{v} = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n)^\top = ([\underline{v_1}, \overline{v_1}], \ldots, [\underline{v_n}, \overline{v_n}])^\top$ denotes the set of vectors $u \in \mathbb{R}^n$ such that $u_i \in \mathbf{v}_i$. The set of n-dimensional real interval vectors is denoted $\mathbb{IR}^n$. An interval vector may be viewed as a hyperrectangle in $\mathbb{R}^n$. We can extend the usual vector operations to interval vectors in the obvious way. Note that the dot-product of two interval vectors is a real interval. We can also consider other interval vector spaces, such as $\mathbb{IR} \times \mathbb{IR}_2$, with entries drawn from $\mathbb{IR}_k$.

When a matrix multiplies an interval vector, the exact result is not necessarily an interval vector, since the entries of the result are not independent. For example, in

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} a\mathbf{x} + b\mathbf{y} \\ c\mathbf{x} + d\mathbf{y} \end{pmatrix}$$

the elements of the resulting vector are not independent. Another way of viewing this phenonmenon is as the fact that a linear transformation of a rectangle may rotate it. What this means is that when we extend matrix-vector multiplication to intervals by $M\mathbf{x} = \Box\{Ma \mid a \in \mathbf{x}\}$, we inevitably introduce overestimation (see Figure A.1). This overestimation can be avoided by using rotated intervals (i.e, representing sets in $\mathbb{R}^n$ as interval vectors plus a linear transformation).

Another source of overestimation arises when we consider interval matrices. An interval matrix $\mathbf{M} \in \mathbb{IR}^{n \times m}$ has entries $\mathbf{m}_{ij} \in \mathbb{IR}$; matrix multiplication is extended in the obvious fashion. This has the effect, as above, of introducing dependencies among the entries of the result, giving rise to overestimation when the hull operator is applied. This is the case whenever interval matrices with mutually dependent entries

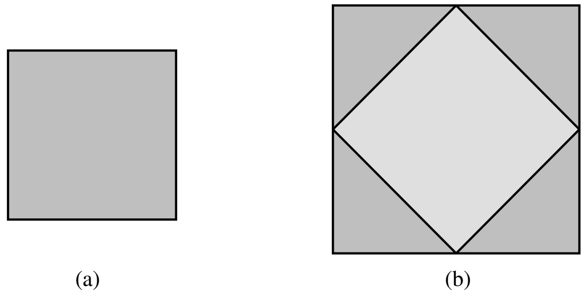


Figure A.1: Rotational overestimation. (a) A 2-dimensional interval vector. (b) The best interval estimate (in dark) after rotating (a) by 45°.

(such as known symmetric matrices) are considered. For special cases, overestimation due to dependence of matrix entries can be dealt with by using extremal analysis as above; we do this for planar rotations in Section 5.3.

A.4 Interval Estimation

Given the methods for dealing with intervals as discussed above, we can now consider how to use intervals as an uncertainty representation for estimation. We formulate the estimation problem for discrete time as follows. We are given a plant with states $x \in \mathbb{R}^n$, evolving according to:

$$x(t+1) = f(x(t), u(t)) + w$$

where $u(t)$ are known inputs and $w \in \mathbf{w}$ is random noise assumed to lie within a known interval $\mathbf{w} \in \mathbb{IR}^n$. Observations $z \in \mathbb{R}^m$ are modeled by:

$$z(t) = g(x(t)) + v$$

where the noise $v \in \mathbf{v}$ is again assumed to lie in a known interval in $\mathbb{IR}^m$. Estimates of $x(t)$ are intervals $\hat{\mathbf{x}}(t)$, approximately representing the set of possible states the system could be in. If point estimates are desired, the (element-wise) midpoint of $\hat{\mathbf{x}}(t)$ could be used (implicitly assuming that noise is unbiased). The initial estimate is $\hat{\mathbf{x}}(0) = [-\infty, \infty]^n$, allowing all possible states. Given an observation $z(t)$, the estimate is updated by:

$$\hat{\mathbf{x}}'(t) = \hat{\mathbf{x}}(t) \cap \mathbf{g}^{-1}(z(t) - \mathbf{v})$$

where $\mathbf{g}^{-1}$ is the interval inverse of g. State extrapolation is simple:

$$\hat{x}(t+1) = \mathbf{f}^*(\hat{\mathbf{x}}'(t), u(t)) + \mathbf{w}$$

where $\mathbf{f}^*$, as above, is the interval extension of f.

It can be shown that if (a) $\mathbf{v}$ and $\mathbf{w}$ are tight bounds on the error and (b) $\mathbf{f}^*$ and $\mathbf{g}^{-1}$ are known exactly, then $\hat{\mathbf{x}}(t)$ converges to $x(t)$ (limited by the driving noise bounds $\mathbf{w}$). If the error bounds are not tight, then uncertainty is introduced proportional to the amount of slack in the bounds. In many cases, the techniques described above in Section A.2 can be used to get exact solutions or good approximations to $\mathbf{f}^*$ and $\mathbf{g}^{-1}$. When g cannot be inverted analytically, numerical techniques can somtimes be used to good effect [Hager, 1990]. A third difficulty that arises in interval estimation is the problem of overestimation, which occurs when elements of the state space are geometrically dependent (as by rotation, for example). For some applications, this overestimation can be tolerated, and for some others, specialized solutions can be applied (such as using rotated intervals). The primary advantages of interval estimators, over statistical estimators like the Kalman filter [Gelb, 1974] is that interval estimators do not make specific statistical assumptions and do not require any linearization.