Probabilistic Navigation

Coping with Uncertainty in Robot Navigation Tasks

Andrew Howard

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

March 1999

Department of Computer Science and Software Engineering

The University of Melbourne

Abstract

Navigation is an important problem for mobile robots. Roughly speaking, this problem can be broken into two parts: localisation and planning. Localisation is the process by which the robot figures out where it is; planning is the process by which the robot figures out how to get to where it wants to be. In attempting to solve the navigation problem, the vast majority of existing systems make a key assumption: that the localisation process can determine the robot's location *unambiguously*. This does not mean that there is no uncertainty in the estimate of the robot's location, merely that there is only *one* such estimate. Unfortunately, this assumption severely limits the robustness of these system: a human operator must provide the robot with an initial estimate of its location, and if this estimate becomes corrupted (for whatever reason), there is no mechanism by which the system can recover. In other words, if the robot gets lost, it stays lost.

This thesis describes a navigation system that is specifically designed to address this problem. Rather than generating a single estimate of the robot's location, this system maintains a *probability distribution* over the space of all *possible* locations. It then analyses this distribution to find a sequence of actions that will not only take the robot to the goal, it will allow the robot to *know* that it has reached the goal. With this system, even if the robot is completely lost, it will still be able to reach the goal.

This thesis describes an *entire* navigation system, all the way from the nuts-and-bolts of the robot hardware, to the high-level planning software. Given a map of the environment, this system is able to guide the robot to the goal, even when the robot's initial location is completely *unknown*.

This is to certify that:

(i) the thesis comprises only my original work,

(ii) due acknowledgment has been made in the text to all other material used,

(iii) the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies, appendices and footnotes.

....

Andrew Howard

Acknowledgments

There are many, many people who have contributed to this thesis in one way or another.

> My supervisor, Les Kitchen, with whom I spent a couple of hundred hours in wide ranging discussions. Without Les' quiet enthusiasm (and administrative sleight-of-hand), this thesis would not have been possible.
>
> The folks at the Vision Lab, and Sandy and Nick in particular, who allowed me waste far too much time in heated debate. Their constructive criticism (and sometimes not-so-constructive criticism) has been invaluable.
>
> The guys in Tech Services, who helped me design and build a number of robots, and in the process, tried to teach me a little of their craft. I would like to thank John, in particular, for meeting my many unreasonable demands with constant good humour.

Finally, and most importantly, I would like to thank Deb, for giving me the confidence to make it happen.

Contents

List of Figures

Chapter 1

Introduction

> "It's a funny thing about Tiggers." whispered Tigger to Roo, "how Tiggers *never* get lost."
>
> "Why don't they, Tigger?"
>
> "They just don't," explained Tigger. "That's how it is."

from *The House at Pooh Corner*, by A. A. Milne.

1.1 Problem Statement

As humans, navigation is a task that we take completely for granted. We drive to the shops, ride to work and walk to the tea-room without ever giving it a second thought. For robots, however, such tasks are extremely difficult.

Why is this?

The navigation problem can be broken down into two parts: localisation, which is the process of figuring out where you are; and planning, which is the process of figuring out how to get to where you want to be. It is the first of these problems that robots find most difficult. To the human eye, the environment is filled with features that enable us to quickly and accurately determine our location. To the impoverished senses of a robot, however, an unmodified human environment is rather like a hedge maze: every part of it looks pretty much like every other part it. Consequently, it is very easy for the robot to become confused and lost.

There are three ways to address this problem:

1. Improve the robot's perceptual abilities to the point where it can employ the same kind of naturally occurring landmarks that humans use. This is the ideal solution, but despite a great deal of effort, it has proved very difficult to mimic the perceptual abilities of human beings.

2. Modify the environment to make it easier for the robot to find landmarks. One can, for example, fill the environment with artificial beacons, or make use of a global positioning system such as satellite GPS. Unfortunately, this solution is not appropriate for all applications.

3. Accept that the robot has impoverished *perceptual* abilities and instead improve its *localisation* and *planning* abilities to the point where it can cope with

uncertainty in its location. In the extreme case, this means figuring out how to reach the goal, even when the robot's initial location is completely unknown.

The first and second of these approaches have been widely explored; the third approach, however, has been largely overlooked (with a few notable exceptions that are discussed in Chapter 9). Consequently, this thesis describes a robot navigation system that employs this latter approach. Given a model of the environment, this system is able to guide the robot to a specified goal, using only naturally occurring landmarks, even when the robot's initial location is *completely unknown*.

1.2 Probabilistic Navigation

While there are a number of different ways to tackle the robot navigation problem, the vast majority of robot systems employ some form of landmark-based navigation. These systems generally break the problem into three processes – modelling, localisation and planning – which are defined as follows:

- Modelling is the process whereby the robot builds a model of its environment. This model will generally contain at least two kinds of information: it will describe the distribution of landmarks within the environment, and it will describe the connectivity of the environment (i.e. how the robot can get from one location to another).

- Localisation is the process whereby the robot infers its location. Usually, it does this by matching the landmarks it sees against those recorded in the model.

- Planning is the process whereby the robot attempts to find a series of actions that will take it from its current location to the goal location. To do this, it makes use of the connectivity information in the model.

Furthermore, the vast majority of robot navigation systems make two key assumptions: that the environment model exists a priori (i.e. that it is provided by a human operator); and that the localisation process can unambiguously determine the robot's location (this does not mean that there is no *uncertainty* in the estimate of the robot's location, merely that there is only *one* such estimate). The landmark-based navigation system described in this thesis retains the first assumption (it does not tackle the issue of model learning) but relaxes the second (it allows for ambiguity in the robot's location). Hence, in this thesis, the processes of modelling, localisation and planning are defined as follows:

- Modelling is the process whereby a human operator provides the robot with a model of its environment. This model describes both the distribution of landmarks and the connectivity of locations.

- Localisation is the process whereby the robot infers the *set of locations* at which it might be. In practice, this means constructing a *probability distribution* over the space of all possible locations.

- Planning is the process whereby the robot tries to find a series of actions that will take the robot from its current location to the goal location, even when there is ambiguity in the robot's location. Furthermore, the planning process must be such that when the robot does eventually reach the goal, there must be no ambiguity in its location. That is, the robot must not only reach the goal, it must know that it has reached the goal.

Unfortunately, as I will show, the generalised planning problem defined above is extremely difficult to solve. Therefore, the navigation system described in this thesis makes use of the concept of *navigation strategies*. A navigation strategy represents a partial solution to the planning problem – one that will enable the robot to reach the goal most of the time, and in most environments. I will describe a number of such navigation strategies in detail in Chapter 9.

There is one further assumption that is made throughout this thesis:

- The environment is (mostly) static.

This assumption is made more for the sake of simplifying the following discussion than out of strict necessity. The navigation system described in this thesis is quite capable of handling some degree of variation in the environment. It is, for example, quite indifferent to the presence of moving obstacles, such as people, unless they actively set out to block the robot's path. Nevertheless, since none of the components of the system were specifically designed to handle dynamic situations, the discussion will focus primarily on static environments.

The robot navigation system described in this thesis is extremely robust: it is very hard for the robot to get lost, and, even if it does, it generally recovers. For example, one experiment I have conducted on many occasions works as follows:

> The robot is taken to some random location within the building in which I work (this location is not known to the robot). The robot is placed on the ground and switched on. If the robot hasn't found its way back to my office within 15 minutes (which it almost always does), I go out and look for it.

This is, of course, a somewhat risky practice that has lead to a couple of unfortunate incidents (on one occasion, the robot caught fire and had been burning for some minutes before it was discovered). It does, however, demonstrate the robustness of the approach.

1.3 Thesis Content

This thesis describes an *entire* robot navigation system, all the way from the nuts-and-bolts of the robot hardware, through to some relatively high-level planning software. Consequently, it contains a number of chapters devoted to topics such as obstacle and landmark detection which, while interesting in their own right, can be seen as preliminary material. The real 'meat' of the thesis is contained in Chapters 7 and 9, both of which look at the navigation problem, but from different perspectives.

The structure of the thesis, like the system it describes, is highly modular. Each chapter describes a particular aspect of the overall system, and can be read as a largely self-contained unit. One consequence of this is that there is no 'Experimental Results' chapter. Instead, each chapter presents experimental results that are appropriate to the content of that chapter.

Finally, note that this thesis *does not* contain a deep theoretical analysis of the generalised navigation problem described above. Nor does it go into great detail exploring all the ways in which this problem *might* be solved. Instead, it concentrates on a single solution and attempts to demonstrate that this solution is both simple and effective.

1.4 Key Contributions

This thesis is heavily influenced by the recent literature on probabilistic navigation, particularly the work of Simmons [224], Koenig [140], and Thun [243] (amongst others). The thesis makes two key contributions to this field:

- It introduces a new method for representing probability distributions, called the *population-based* representation, that has a number of advantages over existing representations.
- It introduces a novel navigation strategy, called the *persistent strategy*, that will function successfully with very high levels of uncertainty.

Both of these innovations are described in detail in Chapter 9. In addition, the thesis treats a problem that has not been widely addressed: how does a robot, in a known environment, but with an unknown initial location, go about reaching a distant goal?

1.5 Thesis Structure

The remainder of this thesis is structured as follows.

- **Chapter 2. Literature Review** reviews some of the contemporary literature on mobile robotics. This chapter is intended to place the work described in this thesis within the wider context of current research.
- **Chapter 3. System Overview** presents a brief overview of the navigation system in its entirety. This chapter serves a 'roadmap' for the remainder of the thesis.

- **Chapter 4. Implementation** briefly describes the hardware and software used to implement the system.

- **Chapter 5. Building Local Maps Using Sonar** describes a novel technique for detecting obstacles and recording their location using sonar.

- **Chapter 6. Building Local Maps Using Vision** describes a novel technique for detecting obstacles and recording their location using vision.

- **Chapter 7. Local Navigation** describes a novel technique for guiding a robot at high speed through a cluttered environment.

- **Chapter 8. Landmark Detection** describes a simple technique for detecting natural landmarks.

- **Chapter 9. Global Navigation** (finally) addresses the generalised navigation task described above.

- **Chapter 10. Conclusion** presents some conclusions and indicates directions for further research.

Chapter 2

Literature Review

> "Where do you come from?" said the Red Queen.
>
> "And where are you going? Look up, speak nicely, and don't twiddle your fingers all the time."

from *Through the Looking Glass*, by Lewis Carroll.

2.1 Introduction

The purpose of this chapter is two-fold: to provide an overview of the current state-of-the-art in mobile robotics, and to place the work described in this thesis within that broader context. As such, the review is intended to be neither detailed nor complete; detailed reviews are given, where appropriate, in each of the following chapters.

The review is divided into five areas:

- Platforms, mobility and applications;
- Sensor technology;
- Control architectures and philosophies;
- Local navigation techniques (i.e. obstacle avoidance);
- Global navigation techniques (i.e. modelling, localisation and planning).

There are, of course, many topics not covered in this review, such as multi-robot systems. Nevertheless, the range of topics presented here is sufficiently broad to provide some context for the remainder of the thesis.

2.2 Platforms, Mobility and Applications

Mobile robots, in one form or another, have entered a very wide range of environments. In this section, I will discuss some of the robots that have been employed on land, under the sea, in the air and in space.

Land

On land, robots come in all sizes, from the very small to the very large. Examples from the small end of the scale include:

- MIT's 'Squirt', which weighs less than 50g and occupies a volume of approximately one cubic inch [92, 91];
- The commercially available, vision-guided 'Khepera' robot, which is smaller than a coffee cup [129].

Examples from the very large end of the scale include:

- Automated cars, trucks and buses [240, 238, 239];
- Load haul vehicles for underground mines [57, 66, 214];
- Cargo container carriers (which weigh up to 80 tonnes when fully loaded) [72];
- Open-cut mining draglines (over 3500 tonnes) [58].

Naturally, most of the robots used for research purposes fall somewhere between these two extremes. For experimental convenience, such robots are sized appropriately for operation in ordinary indoor environments. The three robots that I will describe in this thesis, J Edgar, Piglet and Tigger, all fall into this category. The largest weighs approximately 20kg and stands about 70cm tall; the smallest weighs approximately 1 kg and is less than 20cm in height.

While most of the robots used on land employ wheels of one sort or another (see [46] for a mathematical analysis of some possible wheel arrangements), other methods of locomotion have also been attempted. Most popular is legged robots, with various numbers and arrangements of legs. Some of the combinations that have been attempted include:

- One leg hopping [197];
- Two legs walking [110] and running [194];
- Four legs running [198];
- Five legs walking [123, 252];
- Six legs walking, with various forms of gait (this is by far the most popular configuration) [39, 21, 201];
- Eight legs, with various forms of gait, in applications that include climbing pipes and ducts [177, 206], and rappelling down into volcanos [7, 20].

Various combinations of legs and wheels have also been observed. As for other forms of locomotion, there are snake-like robots [178, 220], and omni-directional robots that make use of spheres instead of wheels [196].

All of the robots described in this thesis use a simple twin-drive-wheel configuration; the kinematics of these robots is discussed in detail in Chapter 4

Sea

Recently, there has been much interest in autonomous underwater vehicles (AUV's). Applications for these robots include mine clearing, exploration and environment monitoring (see [168] for a recent review; see [25] for a collection of papers related to the control of AUV's).

The undersea environment poses quite different mobility problems to those seen on land. On land, robots are generally constrained to move in two-dimensions and have, as a consequence, only three degrees of freedom (position and orientation). AUV's, on the other hand, must move in three dimensions, and consequently have six degrees of freedom (position, plus roll, tilt and yaw). Furthermore, these robots must cope with undersea currents, which tend to push the robot around. In this environment, even the act of station-keeping (i.e. staying in the same spot) poses severe difficulties.

Almost all of the AUV's in use today are neutrally buoyant, and use a set of thrusters for mobility. Some groups, however, have considered the problem from a different perspective, and have set about building robotic *fish* [202, 131]. These robots swim as fish do – using muscular contractions to drive their body through the water. Although much this work was originally intended as a means of understanding the methods of locomotion used by fish, it offers some interesting possibilities for highly manoeuvrable underwater robots.

Air

Autonomous aerial vehicles (AAV) have also received much attention recently. These vehicles are intended for a wide range roles, including surveillance, search-and-rescue missions, search-and-destroy missions, and environment monitoring [168]. AAV's have mobility problems not unlike those experienced by underwater vehicles, except that constraints are more severe – in the air, things tend to go wrong much more quickly, and the robot cannot afford to pause and consider its actions.

Various forms of vehicle have been employed in this context, including planes [115], helicopters [6, 85] (by far the most popular), ducted-fan devices [221] and blimps [256]. Due to its lighter-than-air nature, the latter class is predominantly restricted to indoor applications, where wind speeds are low.

Space

Free-flying robots for space applications have received somewhat less attention than robots intended for other environments. This is natural enough, given the high costs involved in fielding real systems. Most research to date has therefore involved either software or hardware simulation [65, 143]. There are, however, a number of interesting examples of (or proposals for) real systems. Most of these involve free-flying robots that are intended to aid human spaceflight, principally by offering the ability to carry out autonomous inspection or manipulation tasks [111, 146, 167].

For locomotion, these robots utilise the same technologies used in other space vehicles: reaction control systems (otherwise known as rockets) and gyroscopes. The problems faced by these devices are quite different from those faced by robots for land, sea and air applications; in particular, the robot's dynamic properties become extremely important [173]. Once the robot is moving, it is difficult to stop; in addition, when it is interacting with another object, the effect on the robot's own motion can become quite complex and difficult to predict.

Limiting Factors for Robot Design

There are a number of limiting factors that are common to most current mobile robot platforms.

The first limitation is the energy source. Most mobile robots are powered by re-chargeable batteries, which have a comparatively poor energy-to-weight ratio. As a consequence, the duration of these robots tends to be quite limited: it is not uncommon for robots to require recharging every hour (although some of the more recent commercially available robots claim durations of up to eight hours). Duration does, however, tend to be more of a problem for small robots than large ones. Large robots, such as automated cars and mining machines (and even some experimental planetary rovers, such as CMU's Nomad [19]), typically employ internal combustion engines as their principal source of power. For these robots, the parasitic power requirements of computers and sensors are relatively minor when compared to the power needed to drive the vehicle. Consequently, the duration of these robots may be measured in days rather than hours.

The robots described in this thesis will operate continuously for between one and two hours.

The second limiting factor for the current generation of robots is comunications bandwidth. If robots employ some form of off-board processing, or are required to communicate with a human operator at a remote location, bandwidth limitations can be a severe problem. While there are various forms of communication technology that are currently available – such as infra-red devices, ultrasonics and radio – each of these technologies has its difficulties. For example, infra-red devices have a very high bandwidth, but are limited to short range use and line-of-sight operation. Ultrasonic devices (for undersea operation) have very good range, but very poor bandwidth [168]. Radio, which is the most popular technology, also has some problems. In order to sustain high data rates, one must employ high-frequency devices. Unfortunately, with increased frequency comes increased directionality; for long range communication, it becomes necessary to have a directional antenna that can be accurately aimed [18]. In indoor environments, where range is not usually a problem, one observes shadowing and multipath effects. Fortunately, these problems can solved using spread-spectrum technology, which is now broadly available. Off-the-shelf spread spectrum devices with an indoor range of 100m and a data-rate of 85kbps can now be obtained commercially [124].

All of the robots described in this thesis make use of commercially available off-the-shelf radio devices.

To some extent, both of the limitations noted above can be overcome through the use of tethers. However, many, if not most, mobile robot applications preclude the use of tethers (one cannot afford to tether a service robot, for example, or an automated highway vehicle). For this reason, none of the robots described in this thesis employ tethers.

It is worth noting that recent developments in consumer technology, particularly the proliferation of portable devices such as mobile phones and laptop computers, have brought down the cost of small mobile robots and greatly increased their potential capability. These portable devices have exactly the same requirements as small mobile robots: they require much computational power, long battery life and low weight. As a result, the last few years have witnessed the development of powerful but energy efficient processors, such as the Risc-based ARM and StrongArm, and a concurrent improvement in battery technology. Lead-acid and nickel-cadmium cells have evolved into Nickel-Metal-Hydride and Lithium-Ion cells, which have a much higher energy-to-weight ratio.

Two of the three robots described in this thesis, Piglet and Tigger, were designed to exploit these developments in consumer technology. Both these robots are small and very inexpensive. For their size, however, they are quite capable – both are equipped with cameras and are capable of vision-based navigation.

2.3 Sensors

Most of the sensors used on mobile robots have one of two purposes: to detect obstacles, or to determine the robot's location.

Some of the sensors that are used for the former purpose include:

- Simple physical contact sensors, of which there are many variants.

- Infra-red proximity sensors. These units, which are commercially available in many, many forms, are set to trigger whenever an object comes within some pre-set distance.

- Ultrasonic range finders (sonar). These come in various forms, from simple time-of-flight sensors (such as the widely used Polaroid device), to much more complex sensors with greatly enhanced capabilities (such as improved angular resolution and the ability to classify surface types [137]). A number of ultrasonic sensor technologies are reviewed in detail in Chapter 5.

- Laser range finders. These come in two basic varieties: triangulation range finders, in which a source and a detector are placed some distance apart and the distance to an object is determined by triangulation [193]; and time-of-flight range finders [112, 217], in which the source and detector are placed close together, and the distance to an object is determined by the time taken for a laser pulse to travel to the object and back again (in practice, this is often done by measuring the phase difference in an amplitude modulated signal). Time-of-flight laser range finders and their variants have become increasingly popular over the last few years, possibly due to their compact form and increasingly affordable price.

- Radar devices, which work on the same basic principles as laser range finders, have also been used in a number of applications [52, 53, 72]. Until recently, however, their expense has limited their range of applications.

- Various vision-based sensors have been developed. Some of these employ structured lighting [212]; others employ stereoscopic vision, with two or sometimes three cameras [14]. Monocular vision systems, coupled to an appropriate set of image processing algorithms, are also being increasingly used [118, 159]. A number of vision-based obstacle detection techniques are reviewed in detail in Chapter 6.

A more detailed survey of many of these sensors can be found in [126]. Interestingly, while this survey is nearly twenty-years old, it is still largely relevant today.

Other sensors used by robots, mostly to aid in localisation, include:

- Odometry: perhaps the most basic of sensors.

- Inertial navigation systems (INS), which are able to estimate changes in a robot's location by integrating the torques and accelerations experienced by the robot. While INS units are still prohibitively expensive on the whole, they are starting to see applications [175].

- Satellite Global Positioning Systems (GPS), by which the robot is able to obtain an *absolute* measurement of its location. With *differential* GPS, accuracy of less than a meter is possible [87]. The one drawback of GPS is that it tends to work poorly in built-up environments, where buildings can occlude satellites and cause multi-path problems. Nevertheless, for robots operating outside urban environments, GPS is a very useful and relatively inexpensive sensor that has been widely used [107, 147, 185, 232].

While these are the most commonly used sensors, there are many, many alternatives. These include, for example, sensors that make use of active acoustic beacons [136], passive visual beacons [128], passive radar beacons [72], and even *odour* trails, that the robot detects using its sense of *smell* [208, 209].

The three robots described in this thesis are quite impoverished in their sensory capabilities, being limited to odometry and vision. One of the robots (J Edgar) is also equipped with a rotating sonar head.

2.4 Architectures and Philosophies

There have been many architectures put forward to support the construction of intelligent mobile robots [9, 37, 73, 89, 90, 98, 97, 203, 210, 219, 225]. Many of these architectures also embody a particular *philosophy*. Perhaps the best example of this is the Subsumption Architecture, created by Brooks [37, 39, 40, 41]. In contrast with earlier architectures, in which the robot control system was divided into sensing, planning and execution units [179], the Subsumption Architecture decomposes the robot control system into *layers*, each of which expresses a particular robot 'behaviour'. In this approach, successive layers build incrementally on the capabilities of previous layers. For example, at a low level, the robot might have an 'avoid-obstacle' behaviour that prevents it from colliding with obstacles. On top of this one might add a 'wander-at-random' or a 'follow-wall' behaviour. Through the interaction of such behaviours, it is possible to obtain some very sophisticated *emergent* behaviour. The robot Toto is a good example [163]; Toto is able to learn a map of its environment and, on command, return to any given location within that environment. At the same time, the robot is able to avoid unexpected obstacles.

The Subsumption Architecture embodies an entire philosophical approach, both to AI in general and to robotics in particular. While it has had various names, this approach is most commonly known as ‘behaviour-based robotics’. In an early work [39], Brooks identifies four key elements characterising this approach (these four points are quoted directly from [39]):

- [**Situatedness**] The robots are situated in the world — they do not deal with abstract descriptions, but with the here and now of the world directly influencing the behaviour of the system.

- [**Embodiment**] The robots have bodies and experience the world directly – their actions are part of a dynamic with the world and have immediate feedback on their own sensations.

- [**Intelligence**] They are observed to be intelligent — but the source of intelligence is not limited to just the computational engine. It also comes from the situation in the world, the signal transformations within the sensors, and the physical coupling of the robot with the world.

- [**Emergence**] The intelligence of the system emerges from the system's interactions between its components—it is sometimes hard to point to one event or place within the system and say that is why some external action was manifested.

While there has been significant criticism of the behaviour-based approach [249], there is no doubt that the underlying philosophy has had a significant impact on the field of robotics.

In addition to the Subsumption Architecture, there are a number of other approaches worth noting, such as Arkin's Schema-based architecture [9], Rosenblatt's

Distributed Architecture for Mobile Navigation (DAMN) [203], Firby's Reactive Action Packages (RAPS) [89], and Simmon's Task Control Architecture (TCA) [225].

The Schema architecture provides a framework for combining simple behaviours. In early the versions of this architecture [8] each schema is represented as a *potential field* defined over the robot's local environment. For example, an avoid-obstacle schema might have a potential that is inversely proportional to the distance to the nearest sensed-obstacle, while a move-to-goal schema will have potential that is proportional to the distance to the goal. A weighted sum of these fields is used to derive a *net* potential field, which the robot then 'surfs' down. Unfortunately, as with most techniques based on local potential fields, early versions of the Schema architecture suffered from problems of local minima - the robot would tend to get stuck in box-canyons, for example. Later version of this architecture added Schemas with state [15] specifically to address this issue.

A better solution to the problem of combining simple behaviours is described by Rosenblatt [205] with his Distributed Architecture for Mobile Navigation (DAMN). Here, the system is made up of a set of behaviours and a central command arbiter. Each behaviour may vote either for or against an action; the arbiter collates the votes and executes the winning action. A key difference between the DAMN approach and both the Subsumption Architecture and Motor-Schema approaches is that the 'behaviours' in DAMN do not indicate any specific action for the robot to perform. Instead, they are able to vote for a range of actions. This makes the approach both flexible and powerful, as demonstrated by the fact that it has been used extensively to control both on- and off-road vehicles.

The Reactive Action Packages (RAPS) architecture [89, 90] is an attempt to combine reactive behaviour with deliberative planning. In this architecture, the robot is given a library of basic skills or actions, each of which is useful in achieving some short-term goal. These actions can then be combined into more complex structures which represent the solution to more complex, long-range goals. An important aspect of these structures is that they are *situated*; that is, instead of being simple concatenations of primitive actions, they use the sensed world-state to drive the action selection process. The RAPS architecture includes a symbolic planner for generating these structures from the tasks presented to the system.

In a similar vein, the Task Control Architecture (TCA) [225] makes use of many of the ideas embodied in the Subsumption Architecture, whilst still allowing for deliberative, top-down processing. The key idea embodied in this architecture is that, as the robot's control system becomes more complex, one needs a sophisticated mechanism for controlling and coordinating behaviours. This is particularly true if the robot is faced with multiple goals or has limited resources. The TCA provides a mechanism for implementing such controls, based on fairly traditional notions of top-down task-decomposition.

The navigation system described in this thesis does not employ any of the architectures described above. It does, however, draw on a number of lessons that have been learnt from these architectures and their associated philosophies.

- Firstly, the system must be *modular*; it should be possible to add new capabilities to the system with little or no change to the existing modules.

- Secondly, the system must be developed *incrementally*, with each new set of modules building on the capabilities of earlier modules. Hence, in this system, navigation is divided between a number of modules, some of which are responsible for *local* navigation tasks (i.e. obstacle avoidance), while others are responsible for *global* navigation tasks, such as reaching distant goals.

- Finally, the various modules that make up the system must execute *concurrently.* The robot cannot afford to crash into an obstacle because the obstacle avoidance model is waiting for the global planner to work out where to go next.

This software architecture is described in detail in Chapter 4.

2.5 Local Navigation

Most contemporary robotic systems make a distinction between *local* and *global* navigation tasks. Local navigation is principally concerned with the problem of avoiding obstacles, and solutions tend to be both reactive and sensor-based. Global navigation, on the other hand, is concerned with the problems of localisation, path planning and map building. Here, solutions tend to be more deliberative.

A detailed review of *local* navigation techniques is given in Chapter 7; in this chapter, I will simply identify some of the major paradigms.

A very common solution to local navigation problems is to employ some form of artificial *potential field.* This approach, popularised by Khatib [134] and Krogh [149], represents obstacles as imaginary repulsive forces. The goal, if there is one, is represented as an attractive force. By following the resultant (imaginary) force vector, the robot can steer away from obstacles and towards the goal.

There are unfortunately, a number of problems associated with potential fields, not least of which is their tendency to develop local minima, and to cause dangerous oscillations in the motion of fast moving robots [144]. It is for this reason that a number of authors have sought alternative solutions. Two of the most interesting alternatives have been proposed by Borenstein *et al* [31, 251], with their vector field histogram approach (VFH), and Fox *et al* [93, 94], with their dynamic window approach (DWA). Both of these approaches are discussed in more detail in Chapter 7. This chapter also introduces an alternative approach, called *dynamic utility fusion*, which is designed to address many of the same problems.

2.6 Global Navigation

As noted in the Chapter 1, global navigation can be divided into three processes: modelling, localisation and planning. Localisation is the process by which the robot figures out where it is; planning is the process by which the robot figures out how to get to where it wants to be. In order to solve both these problems, the robot needs a model of the environment.

Modelling

There are two basic classes of models used in mobile robot navigation: *metric* models, in which environment features are described in terms of their position with respect to some global coordinate system [44, 244, 259]; and *topological* models, in which environment features are described in terms of their relationships with other environment features [38, 140, 151, 163, 224]. Each class of model has its advantages and its disadvantages: metric models are easy to construct and maintain, but topological models tend to be more robust in the presence of sensory errors. In addition, topological models correspond more closely to a human understanding of the environment. For example, with a metric model, one would specify the robot's location in terms of its coordinates with respect to the global coordinate system; with a topological model, one would say that the robot is 'near the tea-room'.

The system described in this thesis makes use of *metric* models. There are two reasons for this: metric models are easier to work with, and they permit a more accurate representation of the environment. In addition, as Thrun [245, 243] has recently noted, it is relatively easy to super-impose topological models *on top of* metric models; the reverse, however, is much more difficult (although some authors might disagree; see, for example, the work on *spatial semantic hierarchies* by Kuipers *et al* [151]). The problems associated with different kinds of model are discussed in more detail in Chapter 9.

The environment model used by the robot can come from one of two sources: it can be provided by a human operator, or it can be learnt by the robot itself. The problem of model learning has been considered by a number of authors [11, 67, 68, 139, 151, 186, 243, 246, 244, 259, 260]. Model learning is, however, a fundamentally difficult problem: in order to know its location, the robot needs a model of the environment, but in order to generate a model, the robot needs to know its location. Hence this problem is sometimes referred to as SLAM, for simultaneous localisation and mapping.

The problem of model learning is, unfortunately, beyond the scope of this thesis. Instead, it is assumed that a human operator has provided the robot with a model of the environment that is both complete and (reasonably) accurate.

Localisation

Approaches to localisation can be roughly divided into two classes.

- Those that consider localisation as a *static* problem. These approaches attempt to determine the robot's location based entirely on the information that is available to the robot at its *current* position [226, 234, 235, 236]. There is much in common between these techniques and the object pose determination techniques used in computer vision [29].

- Those that consider localisation as a *dynamic* problem. In these approaches, the estimate of the robot's location is based both on the information that is currently available to the robot, and information about the robot's history.

It is the latter class of techniques that are of interest in this thesis.

The vast majority of approaches to dynamic localisation make use of statistical estimation techniques, such as maximum likelihood estimation or Kalman filters [32], to maintain a simple probabilistic estimate of the robot's location [60, 72, 157, 158, 259, 260]. Because they use simple uni-modal distributions (such as the normal distribution), these techniques are quite limited. They require a good initial estimate of the robot's location, and if this estimate becomes corrupted for some reason, there is no mechanism by which the robot can recover. In other words, if the robot gets lost, it stays lost.

Of more interest in this thesis are *probabilistic* localisation techniques [44, 130, 140, 224, 243, 246, 244], in which the robot maintains a *probability distribution* over the space of all possible robot locations.[1] This approach offers a much more robust solution, since it does not require a good initial estimate of the robot's location. Consequently, if the robot gets lost, it is possible to recover. In fact, as I will show in Chapter 9, it is entirely possible for a robot to reach its goal, even when its initial location is completely unknown.

[1] These techniques are often referred to as *Markov* localisation, since they can be derived from the theory of Markov Random Fields [86]. The same results can, however, be derived from first principles using simple Bayesian analysis. Consequently, in this thesis, the term *probabilistic* localisation is used.

Planning

There is a vast body of literature devoted to the problem of path planning for mobile robots. Traditional methods include applying A* search to visibility graphs formed in configuration space [161, 200]. Techniques involving distance transforms [127, 265] and harmonic functions [264] have also been employed. All of these approaches assume that the robot's location is known, at least approximately. Unfortunately, when using a probabilistic localisation technique, it is not always possible to make this determination. Sometimes, the best one can do is indicate that the robot may be at one of a *number* of possible locations. Consequently, in this context, the applicability of these traditional methods is far from clear.

The few authors that have considered the problem of planning in the context of probabilistic localisation [130, 140] all take more-or-less the same approach as that described in this thesis – the use of navigation *strategies* or *policies*. While these strategies are known to be sub-optimal, they do at least represent a practical solution to an otherwise intractable problem.

Navigation strategies are considered in detail in Chapter 9, where I will also introduce a novel strategy that has certain advantages over those that have been described previously.

2.7 Conclusion

The navigation system described in this thesis has been designed for operation in indoor environments, using small robots with simple sensors. The reasons for this are mostly pragmatic: there is a broad range of applications for robots operating in indoor environments, and this is the environment that is most readily accessible for experimental purposes. Nevertheless, it is hoped that many of the ideas expressed in this thesis will also find application in more hostile environments, such as underground mines and construction sites.

Chapter 3

System Overview

> 'What have you done to that security robot?'
>
> The little robot was rolling very, very slowly round the ceiling and moaning quietly to itself.
>
> 'I've made it very happy,' snapped Ford. 'It's a kind of mission I have...'.

from *Mostly Harmless*, by Douglas Adams.

Abstract

This chapter serves two purposes: to give a broad overview of the navigation system described in this thesis, and to highlight some of the more interesting aspects of this system. Consequently, it should be seen as both a 'roadmap' and a preview for the remainder of the thesis.

3.1 Introduction

The navigation system described in this thesis divides the overall system functionality between a number of system *layers*, each of which is responsible for a specific set of tasks. There are four such layers:

- The **hardware abstraction layer** acts as an interface between the physical robot hardware and the rest of the system. This layer presents an *abstract* interface to other system layers, allowing the system to operate transparently on a number of different robot platforms (including simulated robots).

- The **local layer** is responsible for carrying out *local* navigation tasks, i.e. for detecting and avoiding obstacles whilst reaching nearby goals.

- The **global layer** is responsible for global navigation tasks; i.e. for reaching distant goals. This layer does not control the robot directly; instead, it sets nearby goals for the local layer.

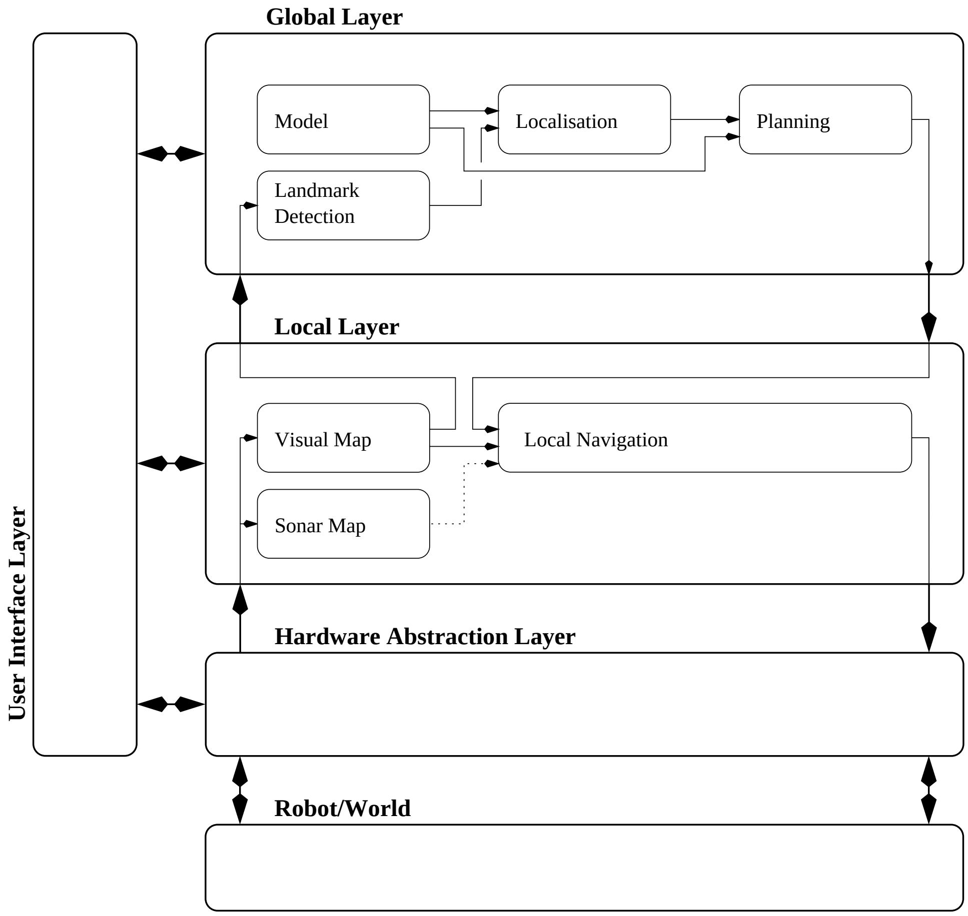


Figure 3.1: System diagram showing the hardware abstraction layer, the local layer and the global layer. The arrows indicate the flow of information around the system (some arrows have been omitted for the sake of clarity).

- The **user interface layer** is responsible for managing the user interface on the host computer. A good user interface greatly accelerates the software development process and ultimately leads to a more robust system.

Layers are further broken down into *modules*, each of which is responsible for a specific subtask. Thus, for example, the local layer has modules for map building and navigation, while the global layer has modules for landmark detection, modelling, localisation and planning. The relationship between these modules is indicated in Figure 3.1, where the arrows indicate the flow of information around the system. Communication between modules is via a message passing mechanism.

A key aspect of this design is the way in which the overall navigation task is split between the local and global layers: the local layer is responsible detecting and avoiding obstacles, whilst the global layer is responsible for reaching distant goals. This division of responsibilities has a double benefit: local navigation can be treated as a purely reactive problem, in which no long term planning is required; and global navigation can be treated as a fairly abstract planning problem, since all the messy details of actually detecting and avoiding obstacles can be delegated to the local layer.

In the following sections, I will briefly outline the function of each of the four system layers, and of the modules from which the are composed.

3.2 The Hardware Abstraction Layer

The hardware abstraction layer provides an abstract interface through which the rest of the system can interact with a variety of robot platforms. In the course of this research, I have in fact used three different robots: J Edgar, Piglet and Tigger (described in the following chapter). The hardware abstraction layer provides a common interface to all three robots. When required, it also serves as a generic robot simulator.

The hardware abstraction layer supports a range of functionality. Through this layer, one can:

- Specify the robot's linear and angular velocity;
- Estimate the robot's position using odometry;
- Point the robot's sonar head in a particular direction;
- Acquire range measurements from the sonar sensor;
- Point the robot's camera in a particular direction;
- Acquire images from the robot's camera.

Different robots have different capabilities, of course, so not all functions are supported by all robots. For example, of the three robots I have used, only J Edgar is equipped with sonar.

Use of a hardware abstraction layer greatly enhances the portability of the overall system. In principle, the system can be ported to a new hardware platform by simply writing a new implementation of this layer. In practice, however, it is not always this simple – the fact that robots share a common interface does not automatically make them interchangeable as far as the rest of the system is concerned. The differing capabilities of the robots – particularly their size and speed – must be taken into account. Consider, for example, the differences between J Edgar and Tigger. Whereas J Edgar has a diameter of about 50 cm and weighs 20 kg, Tigger has a diameter of only 15 cm and weighs 1 kg. Obviously, it is much more difficult for J Edgar to negotiate a cluttered environment at high speed than it is for Tigger, and the modules responsible for obstacle avoidance must take this into account.

3.3 The Local Layer

The local layer has two basic responsibilities: to build maps of the robot's local environment (i.e. the area within a few meters of the robot), and to carry out local navigation tasks. These responsibilities are divided between three *modules*, as follows:

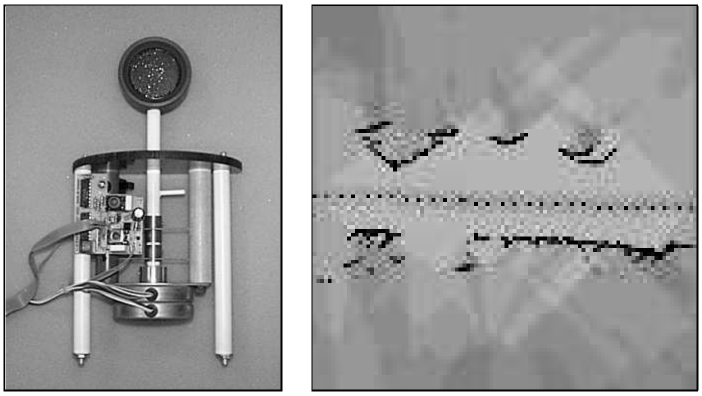
- The sonar mapping module uses sonar range data to build a local occupancy map [77];

- The visual mapping module uses visual range data to build a local occupancy map;

- The local navigation module uses the data from the mapping modules to detect and avoid obstacles, whilst simultaneously attempting to reach a nearby goal.

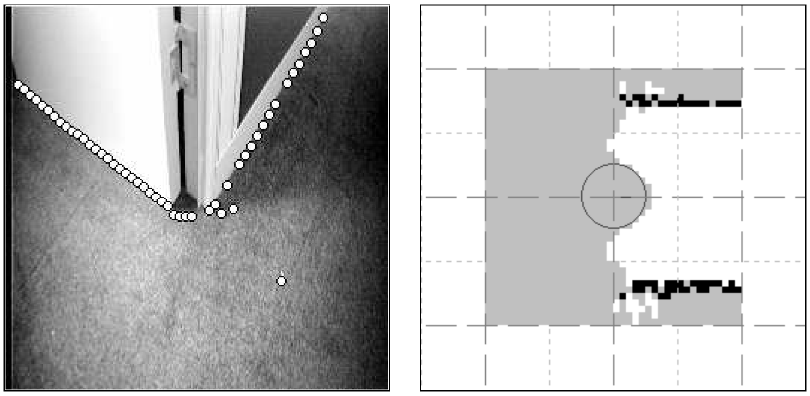
Note that the division into discrete sonar and visual mapping modules is the result of an historical accident, rather than a considered approach. A better way of modularising the system might be to have a common mapping module that supports two 'virtual' range sensors.

In the following sections, I will briefly describe each of the modules listed above.



(a) (b)

Figure 3.2: (a) J Edgar's rotating sonar head. (b) Local cccupancy map produced using sonar. Dark pixels represent occupied cells, light pixel represent unoccupied cells, grey pixels indicate unknown cells.



(a) (b)

Figure 3.3: (a) Image taken by the robot's camera. The white dots denote the boundary of the detected obstacles. (b) A typical occupancy map. Dark pixels represent occupied cells, light pixel represent unoccupied cells, grey pixels indicate unknown cells.

3.3.1 Building Local Maps using Sonar

The sonar mapping module uses sonar range data to construct a *local occupancy map*. This is a simple grid-based map in which each cell may be either occupied (meaning that there is an obstacle at this location) or empty (meaning that there is no obstacle at this location) [77]. Figure 3.2 shows a typical occupancy map. In this figure, black pixels indicate occupied cells, white pixels indicate unoccupied cells and grey pixels indicate cells whose occupancy state is unknown.

The sonar mapping module does not attempt to maintain a map of the entire environment; instead, it maintains a map of the area within a few meters of the robot. There are a number of good reasons for doing this:

- The map building process makes use of odometric position estimates. These estimates are prone to cumulative drift, which makes it difficult to build large maps. If the maps are entirely local, however, odometric drift can be largely ignored.

- The environment is not entirely static: objects may move, doors may open and close. The robot can only detect such changes when they occur within the range of its sensors.

- As far as local navigation is concerned, the state of the environment more than a few meters from the robot is completely irrelevant. It is the role of the global layer to concern itself with such global issues.

Superficially, sonar is an attractive sensor modality for building local maps, mainly because it is low in cost, has low power requirements and is relatively simple to use. Unfortunately, sonar is also has a number of disadvantages:

- It has limited angular resolution. A standard sonar sensor cannot resolve objects that are within about 30° of each other.

- It is prone to specular reflection effects. For sonar, many surfaces act like mirrors, reflecting the emitted sound waves away from the sensor. Such surfaces will be effectively invisible to the sonar sensor. Alternatively, these surfaces may give rise to *multiple reflections*, which result in mis-leading range measurements.

The first of these problems can be partially overcome by combining range measurements from multiple viewpoints; Elfes [76, 77] describes a technique for doing this (the occupancy grid approach) that employs a Bayesian data fusion method. Unfortunately, this technique does not address the problem of specular reflection. The sonar mapping module therefore makes use of a novel technique called the 'response grid'. The basic notion encapsulated by the response grid is that a cell may appear to be occupied (i.e. will generate a response) when viewed from one direction, but may appear to be unoccupied (i.e. will not generate a response) when viewed from another.

Although it is not central to this thesis, a description of the response grid, and the sonar mapping module through which it is implemented, is presented in Chapter 5.

3.3.2 Building Local Maps using Vision

Like the sonar mapping module, the visual mapping module constructs a local occupancy map. However, this module determines the location of nearby objects by analysing the images taken by the robot's camera. To simplify this task, the visual mapping module makes a number of assumptions about the environment the robot inhabits. Specifically, it is assumed that:

- The robot operates in an environment with a well defined ground plane (i.e. an indoor environment with level surfaces).

- The ground is covered by a carpet (or similar floor covering) such that all carpeted areas can be treated as free space and all non-carpeted areas can be treated as obstacles.

- The carpet has a relatively even texture.

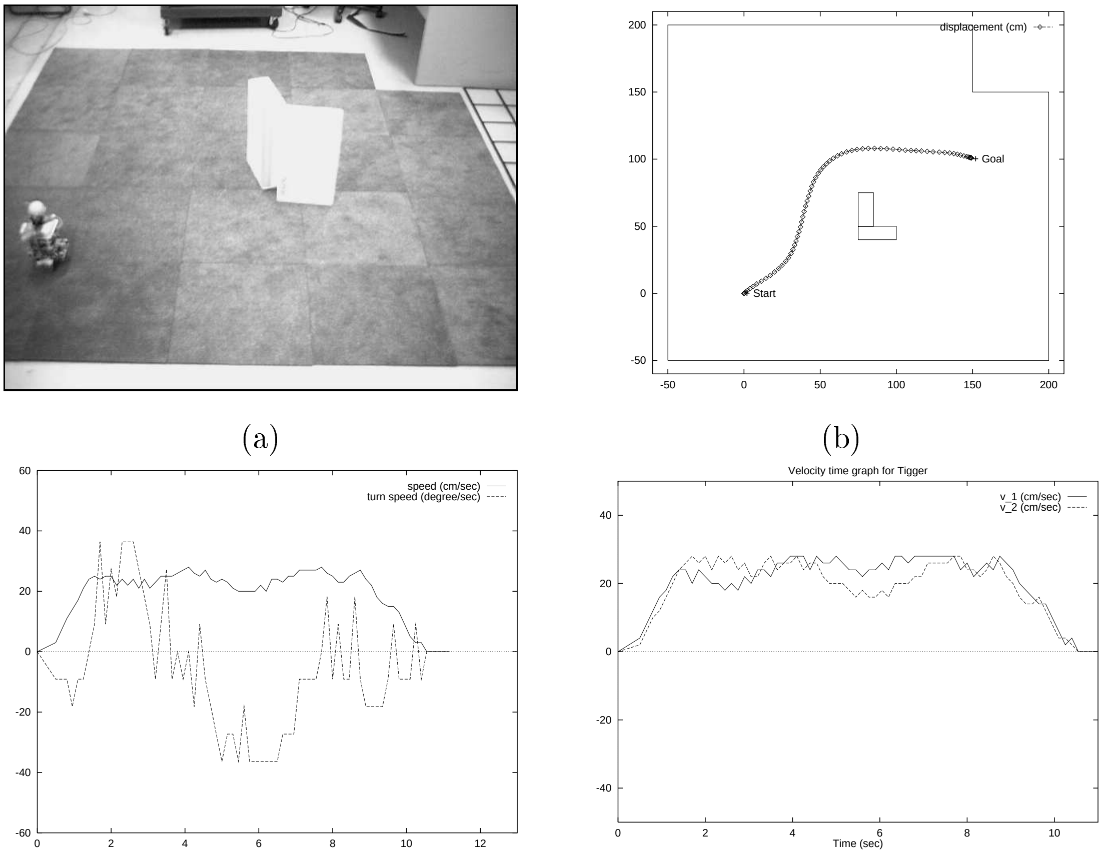
Collectively, these assumptions reduce the general problem of locating obstacles in an image to the much more tractable problem of finding the boundary of an untextured region. Figure 3.3 shows a typical image taken by the robot's camera, together with the occupancy map produced from a series of such images.

After the initial experiments with *sonar* described in Chapter 5, it was decided to use *vision* as the robot's principle sensor. The reasons for this are simple: the visual mapping module is faster and more accurate that the sonar mapping module. These advantages more than compensate for the fact that visual mapping module requires more CPU power than the sonar mapping module.

The visual mapping module is described in detail in Chapter 6.

3.3.3 Local Navigation

The local navigation module is responsible for local navigation tasks; i.e. for reaching nearby goals whilst simultaneously avoiding obstacles. Furthermore, it must enable the robot to travel at a relatively high speed (up to 50cm/sec, or 1.8 km/hr) through



(a) (b)

(c) (d)

Figure 3.4: (a) Experimental setup for testing local navigation. (b) Path taken by the robot. (c, d) Velocity-time graph for the same task.

a cluttered environment. There are a number of standard techniques that can be applied to this kind of problem, such as artificial potential fields [9, 134] and distance transforms [127, 265]. Unfortunately, while these techniques work very well at low speeds, they become increasingly unreliable or unstable as the robot's speed increases [144]. The problem with all of these techniques is that they fail to explicitly account for the dynamics of the robot, i.e. the fact that the robot is capable of only finite accelerations. Consequently, the local navigation module employs a novel technique in which the dynamics of the robot are considered in an explicit fashion.

The local navigation module has three components:

- A set of *constraints*, which determine the *sequences of actions* are physically possible. For example, the robot has a maximum speed and a maximum acceleration, both of which are expressed as constraints.

- A set of *behaviours*, which determine the *utility* of different sequences of actions. For example, sequences that avoid obstacles have a high utility, but so do sequences that move the robot closer to the goal.

- An arbiter, which seeks to find the most desirable sequence of actions that satisfies all of the constraints.

A crucial aspect of this technique is that the arbiter does not look for a single action for the robot to perform; rather, it looks for a *sequence* of actions. To see why this is necessary, consider a robot which is travelling at high speed towards a wall. If the robot is travelling at 100 cm/sec and is capable of decelerating at a rate of 50 cm/sec/sec, the robot must start to decelerate when it is still 100 cm from the wall. If it does not do so, a collision is inevitable. Hence, for a robot travelling at high speed, the finite acceleration of the robot becomes an important factor and the arbiter *must* plan ahead. The arbiter can do this in a number of ways: it can consider either a long sequence of short actions, or a short sequence of long actions, or anything in between. For example, in the situation described above, the robot needs to plan at least 2 sec in advance. It can do this by selecting a sequence of 20 actions each lasting 100 ms, or by selecting a single action lasting 2 sec. The arbiter used by the local navigation module takes the former approach, since it allows the robot to select from a much larger set of trajectories. For a detailed theoretical discussion of the issues involved in high speed navigation, see Kelly *et al* [132].

The motion produced by the local navigation is both smooth and fast: the robot will decelerate into corners and accelerate out of them, it will swerve to avoid collisions, and it will decelerate smoothly to a stop at the goal. Figure 3.4 shows an example of the local navigation module in action: the first graph shows the path taken by the robot; the second shows the robot's linear and angular velocities as a function of time. The path is smooth in position space, and relatively smooth in velocity space. Note that the noise that is apparent in the angular velocity plot is a result of measurement noise, rather than noise in the robot's actual motion. Very short integration times (of the order of 100ms), combined with relatively coarse odometric sensors (the robot will travel about 1.2mm for every 'click'), leads to a great deal of uncertainty in the measured angular velocity. By contrast, if one looks at the individual *wheel* speeds (Figure 3.4d), the motion is quite smooth.

The local navigation module is described in detail in Chapter 7.

3.4 The Global Layer

The global layer is responsible for global navigation tasks. That is, it is responsible for determining the robot's location and guiding it to a distant goal. There are four modules in the global layer.

- The *landmark detection module* uses the local occupancy maps produced by the sonar and visual mapping modules to detect significant landmarks, such as walls and doorways.

- The *modelling module* encodes a model of the environment the robot inhabits. This model describes both the distribution of landmarks within the environment (for localisation) and the connectivity of the environment (for planning).

- The *localisation module* compares the landmarks detected by the landmark detection module against with landmarks stored in the model to construct a probability distribution over the space of all possible robot locations.

- The *planning module* analyses this probability distribution and attempts to find a series of actions such that the robot will not only reach the goal, it will *know* that it has reached the goal.

In the following sections, I will briefly describe each of these modules.

3.4.1 Landmark Detection

The landmark detection module extracts significant features – such as walls, corners and doorways – from the local occupancy map. The procedure used by this module is very simple: it has a set of predefined templates which it tries to match

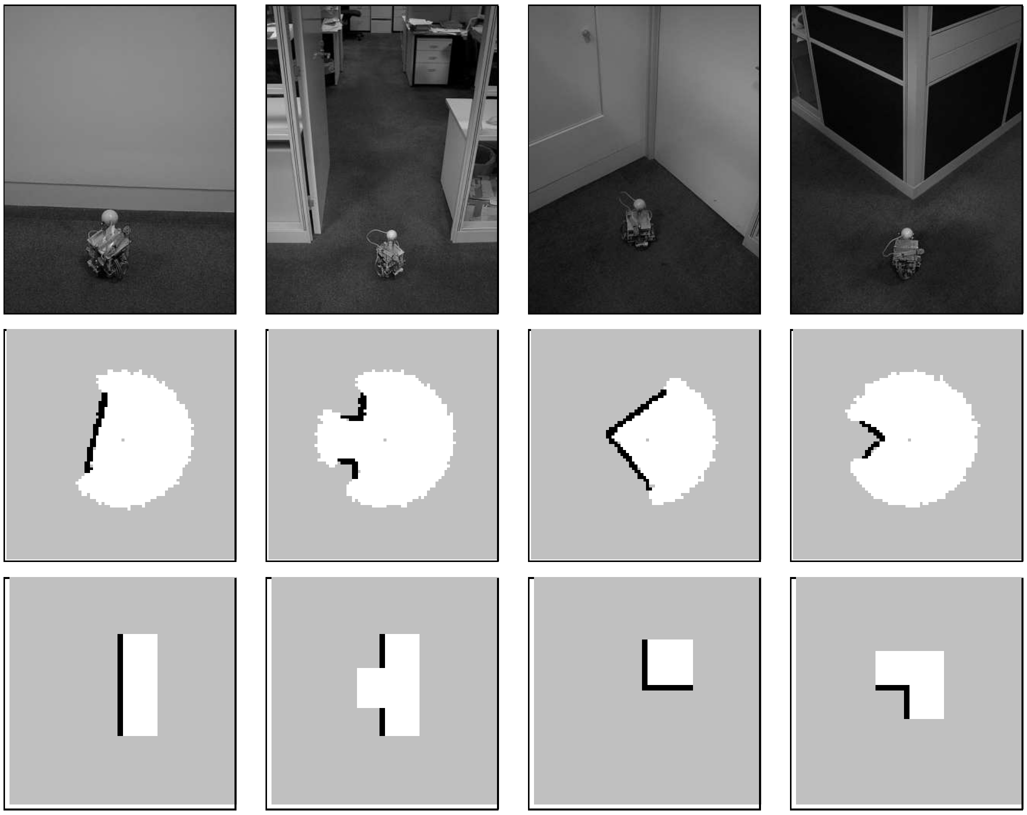


Figure 3.5: A typical set of landmarks. The top row shows the robot Tigger next to: a wall, a doorway, an inside corner, an outside corner. The middle row shows a corresponding set of local occupancy maps – dark cells in these maps indicate regions that are probably occupied, light cells indicate regions that are probably unoccupied, and grey cells indicate regions whose occupancy state is unknown. The bottom row shows a set of templates – dark cells indicate regions that should be occupied, light cells indicate regions that should not be occupied and grey cells indicate regions whose occupancy state is irrelevant.

against the local occupancy map. Empirically, it has been found that a small set of relatively abstract templates can capture useful information about the environment. For example, there is no single template corresponding to a ‘corridor intersection’; rather, each particular intersection can be represented by a collection of simpler landmarks, such as walls and corners.

Figure 3.5 shows a typical set of templates; there are templates here for walls, doorways and corners. The dark pixels in these template correspond to regions that should be occupied, the light pixels to regions that should be unoccupied, and the grey pixels to regions whose occupancy state is irrelevant (i.e. to ‘don’t care’ regions). To locate a landmark, the landmark detection module places a template over the local occupancy map at a randomly chosen position and orientation and evaluates the match between the template and the occupancy map. The match is best when occupied pixels in the template correspond to occupied pixels in the occupancy map, and when unoccupied pixels in the template correspond to unoccupied pixels in the occupancy map. A simple stochastic search process is used to find the best matches. When a landmark is located, information about this landmark is sent to the global localisation module, where it is used to help determine the robot’s location.

The landmark detection module is described in detail in Chapter 8.

3.4.2 The Model

In constructing a model of the environment, one must consider the processes in which the model will be used. In the global layer, there are two such processes: localisation and planning. For localisation, one requires a model that describes the environment in terms of the location of significant landmarks, such as doorways. For planning, one needs a model that describes the connectivity of the environment; i.e. how the robot can move from one part of the environment to another. In addition, the model must somehow capture the uncertainty associated with both sensing and action.

The global modelling module encodes a model made up of a triplet of *probability distribution functions*. The first distribution, called the *unary sensor model*, describes the arrangement of landmarks within the environment, i.e. it describes the probability of observing a particular type of landmark at a particular location. The second distribution, called the *binary sensor model*, describes the properties of the robot's odometric sensors, i.e. the rate at which these sensors drift. The final distribution, called the *action model* describes the connectivity of the environment, i.e. it describes how a robot at one location may go about reaching another.

Note that this model is entirely *metric*; that is, it is defined with respect to a global coordinate system. In Chapter 9, where the global model is described in detail, I will discuss some of the reasons for choosing a metric model rather than a topological one.

3.4.3 Localisation

The localisation module is responsible for determining the robot's location with respect to the global model. To do this, it matches observed landmarks against the model to construct a probability distribution over the space of all possible robot locations. This distribution is not static, of course; it changes over time as the robot moves through the environment and new landmarks are detected. In general, there are two competing forces at work on the distribution:

- The detection of landmarks will tend to concentrate the distribution, i.e. it

Figure 3.6: Evolution of the location distribution in an environment containing wall and corner landmarks. Dark regions indicated locations at which the robot is more likely to be found. The robot's initial location is unknown.

will reduce the number of possible robot locations.

- Motion of the robot will tend to blur the distribution, i.e. it will increase the number of possible robot locations. Robot motion is measured using odometry, which has certain inaccuracies associated with it. Over time, these inaccuracies accumulate, making the location of the robot more uncertain.

As a concrete example, consider the series of images shown in Figure 3.6. These images show the evolution of the probability distribution as the robot moves through an environment containing landmarks such as walls and corners. In this example, the robot's initial location is assumed to be *unknown*; hence the initial distribution is uniform over the entire space. However, as the robot moves, the detection of a number of landmarks quickly reduces the uncertainty in the robot's location.

The foremost challenge in the design of the localisation module is finding an efficient method for representing and updating the probability distribution. The most obvious solution is to divide up the space of possible robot locations into discrete cells, each of which has a probability associated with it. However, as I will show in Chapter 9, this approach has a number difficulties associated with it. As an alternative, I have developed a *population-based* approach that makes use of a genetic algorithm (of sorts). This approach is discussed in detail in Chapter 9, where I will show that it is both efficient and effective.

3.4.4 Planning

The planning module is responsible for finding a series of actions that will not only take the robot to the goal, it will allow the robot to *know* that it has reached to goal. This statement can be made more precise if we describe the robot's state not in terms of its location, but in terms of its *knowledge* about its location. That is, the robot's state is described by the location probability distribution. Consequently, the role of the planning module is to find a series of actions that will take the robot from its initial state (in which the robot's location may be completely unknown) to the goal state (in which the robot is certain that it is at the goal location). In principle, this can be done by simply searching the *space* of such states, using the global model to determine how states are connected. In practice, this is utterly impossible : the search space is huge, even for the most trivial examples.

As an alternative, the planning module make use of the concept of a *navigation strategy*. Navigation strategies represent a partial solution to the planning problem, one that will enable to robot to reach the goal in a reasonable amount of time, on most occasions and in most environments. In Chapter 9, I will describe a number of different navigation strategies and compare their performance in an experimental context.

3.5 The User Interface Layer

The user interface layer implements an extensive and sophisticated user interface. User interface design is an aspect of robotics that is often overlooked – after all, the 'user' is typically the researcher who developed the system, who presumably does not need a sophisticated interface though which to interact with the robot. This argument, however, misses the point. Without an effective user interface, the robot becomes a 'black box' – one can observe its behaviour, but can only guess at the reasoning processes that lead to that behaviour. My personal experience has been that a good user interface, through which the researcher can observe and modify the

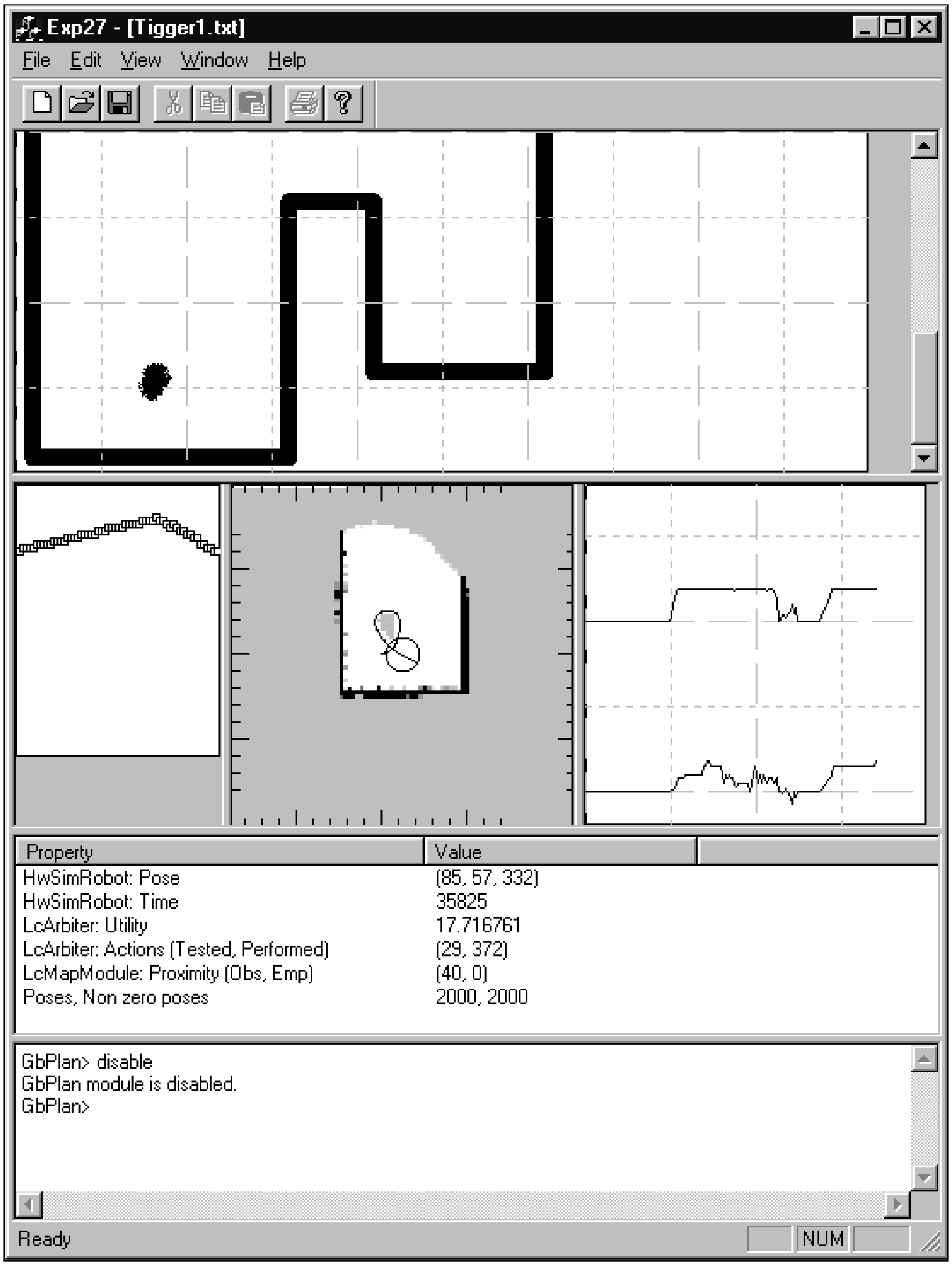


Figure 3.7: Snapshot of the user interface (this is the Windows NT version). Note the various of windows, each of which displays a different kind of information. From top to bottom: the global window, the image window, the local window, the trace window, the status window, the command window.

internal state of the robot, greatly accelerates the software development process. In the long run, it also leads to the creation of a much more robust system.

As a concrete example, consider the user interface shown in Figure 3.7. This figure shows an application with a number of windows, each of which displays a different kind of data. For example:

- The image window shows the raw data coming in from the camera – this allows one to see if the camera is functioning incorrectly or if the lighting conditions are unsuitable.

- The trace window shows the evolution over time of a number of variables, including the robot's speed. This is very useful for detecting a number of problems, such as flat batteries or poorly tuned PID control loops on the robot's motors.

- The local window shows the local occupancy map and the robot's intended path – this allows one to see if the sonar or visual mapping modules are malfunctioning, or if the local navigation module is finding bad paths for the robot to follow.

- The global window shows the global model of the environment, with the probability distribution representing the robot's current belief about its location superimposed.

Some windows can also be used to alter the state of modules. For example:

- The command window allows the user to enter text commands. One can, for example, disable the local and global layers and take direct control of the robot's motors.

- The local window allows the user to specify a goal for the local navigation module (after the global layer has been disabled) by clicking on the map with a mouse. This is extremely useful for testing the performance of the local layer in isolation.

3.6 Conclusion

In the following chapters, each of the modules that make up the local and global layers is described in detail. In addition, Chapter 4 describes the low-level hardware and software on which the system is implemented.

Note that the ordering of these chapters is pedagogical: each chapter describes a set of capabilities that build on those described in previous chapters. The ordering is also largely chronological, reflecting the order in which the research was actually conducted.

Chapter 4

Implementation

> "And now for a drop of tenderness, but I must add only a touch;
> for without a drop of tenderness, he might destroy me...
> whoops, too much!"

Theme song from *Milton the Monster*.

Abstract

This chapter describes the robot platforms on which the navigation system described in this thesis has been implemented. It describes both hardware and software architectures, and lays the groundwork for the chapters which follow.

4.1 Hardware

In the course of this research, I have made use of three different mobile robot platforms: J Edgar, Piglet and Tigger (see Figure 4.1). Two of these robots – Piglet and Tigger – were built from scratch, while J Edgar is a heavily modified 'Maxi-Fander' robot from Branch Denning Associates. In the following sections, I will briefly describe each of these robots: their capabilities, strengths and weaknesses. I will also look at the issue of robot kinematics, which is common to all three platforms.

4.1.1 J Edgar

J Edgar started its life as a 'Maxi-Fander' robot from Branch Denning Associates, and is so named because in its original incarnation it bore a strong resemblance to a vacuum cleaner. Since then, I have rebuilt this robot a number of times to turn it into a more useful research platform. J Edgar's specifications are given in Table 4.1. Basically, this robot can be thought of as a PC on wheels, which is further equipped with a rotating sonar head, a camera and frame-grabber.

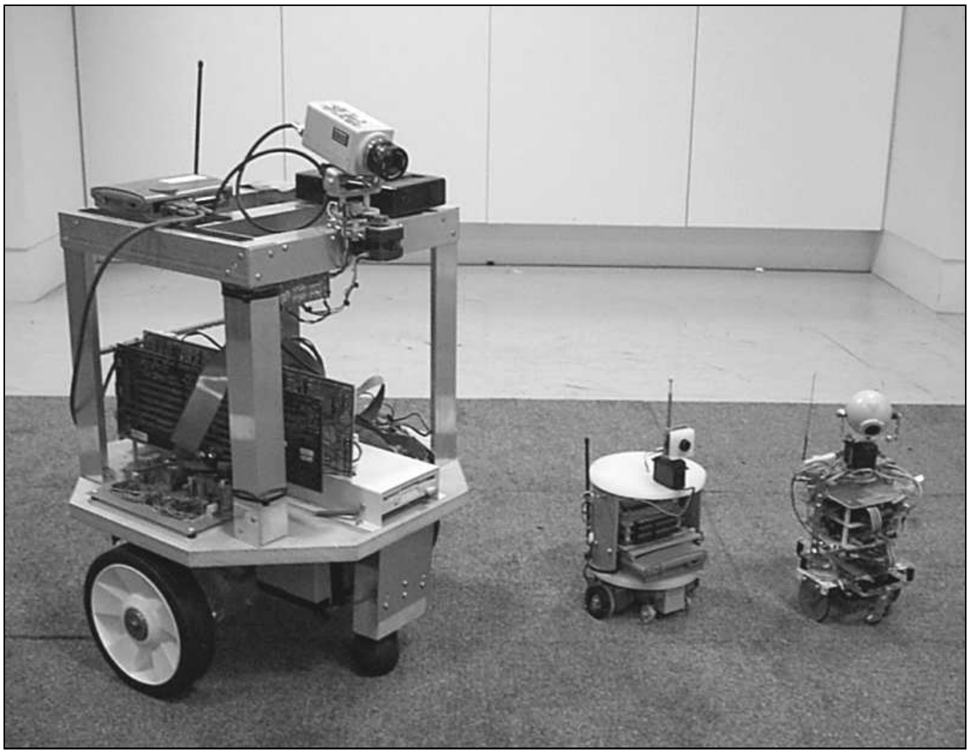


Figure 4.1: Family portrait, from left to right: J Edgar, Piglet and Tigger

An important aspect of J Edgar's design is that it uses a mixture of on-board and off-board processing. The on-board processor is connected to an off-board processor (in the form of a PC or workstation) via a radio modem. Most computation is carried out off-board, leaving the on-board processor only the most basic of tasks. This general configuration is shown in Figure 4.2. There are a number advantages to this arrangement:

- Power – the on-board processor is one of the major drains on the robot's batteries, and hence is one of the limiting factors for battery life. Reducing the amount of processing the robot has to carry on-board therefore increases battery life.
- Cost – off-board computation is much cheaper than on-board computation. While the off-board processor can be an existing PC or workstation, the on-board processor must generally be a dedicated device.
- Ease of development – it is much easier to develop code for the off-board processor (where standard interactive debugging tools can be used) than it is to develop code for the on-board processor (where one must employ program traces, remote debuggers and the like).

Note that there are some functions which cannot be carried out off-board. In particular, the high latency and relatively low data-rates of the radio modem means that both low-level motor control and image processing must be carried out by the on-board processor.

From a research perspective, J Edgar has a number of limitations: it is large, it is heavy and it has quite limited battery life (about 1.5 hours of continuous operation). The size, in particular, is a limiting factor – it is difficult to find the space necessary to construct interesting environments in which to carry out experiments. Most experiments conducted using J Edgar have therefore been carried out within the corridors of the building in which I work. While this has been useful in 'keeping me honest' (since I don't have the opportunity to 'tweak' the environment to suit the robot), it has also made it difficult to test certain elements of the system's behaviour. The robots Piglet and Tigger, which are much smaller than J Edgar, where built specifically to address this limitation.

4.1.2 Piglet

Piglet (which is named in honour of the A. A. Milne character from *Winnie-the-Pooh*) is a bare-bones robot, which takes the notion that the robot's on-board processor should do as little as possible to the extreme. This robot has practically zero on-board intelligence. It also has no means for acquiring and digitising images. Instead, raw video from the robot's camera is fed into an analogue video transmitter, from which it is sent back to a base station and into a frame-grabber connected to the off-board processor. The general configuration is shown in Figure 4.3. Piglet's specifications are given in Table 4.2. Piglet can be thought of as a remote control vehicle with a camera strapped to it.

Piglet has many advantages over J Edgar: it is small, it is light and it costs less than one-tenth the price of J Edgar. However, it also has a number of significant limitations:

- The quality of the video transmitter's signal is highly variable. As the robot moves, the image may be obscured by static or may disappear completely. Also, the video transmitter and radio modem (both of which are RF devices) tend to interfere with one another.

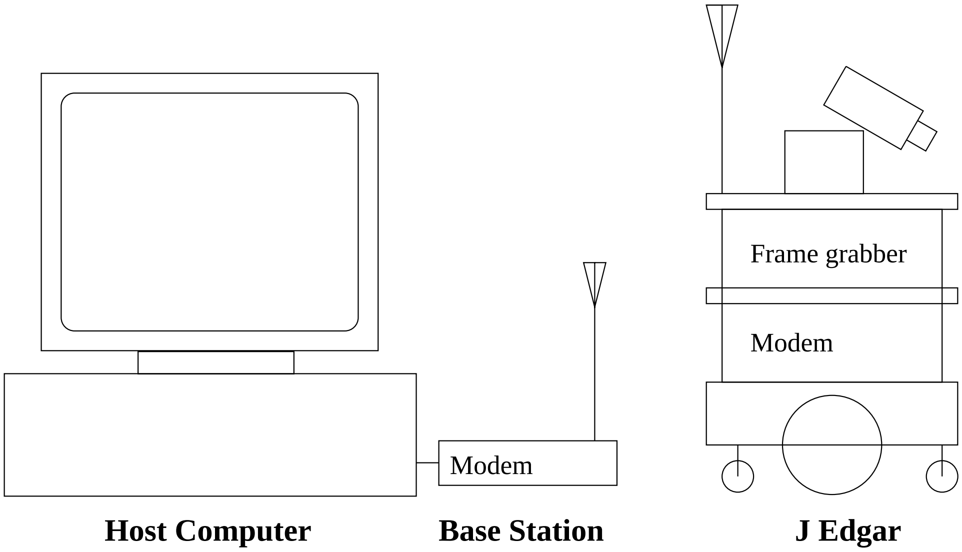


Figure 4.2: Physical setup for J Edgar. Processing is divided between the host computer and the robot's on-board CPU. Communication is via a radio modem. Images from the camera are digitised by the on-board frame-grabber.

CPU	Standard PC motherboard with 486 CPU running at 33 MHz.
Sonar	Single sonar transducer mounted on a rotating head
Vision	Monochrome CCD camera, frame-grabber
Comms	Spread-spectrum radio modem (maximum throughput 85kb/s)
Programming	Via floppy disk
Kinematics	Twin drive wheel (non-holonomic)
Dimensions	50×65 cm (w $\times$ h)
Weight	20kg
Battery life	1 hr (approx)

Table 4.1: Specifications for robot J Edgar.

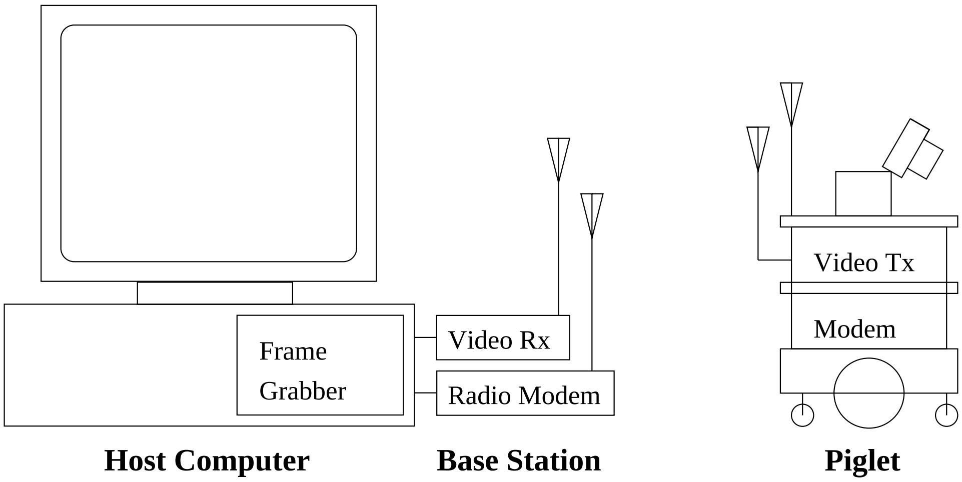


Figure 4.3: Physical setup for Piglet. Raw video from the camera is passed to an analogue video transmitter, from which it is sent back to the base station.

CPU	Microcontroller (Microchip PIC)
Sonar	None
Vision	Monochrome CCD PCB camera, analogue video transmitter
Comms	Spread-spectrum radio modem (max throughput 85kb/s)
Programming	Via EPROM
Kinematics	Twin drive wheel (non-holonomic)
Dimensions	15 $\times$ 25 cm (w $\times$ h)
Weight	1kg
Battery life	1 hr

Table 4.2: Specifications for robot Piglet.

- The robot employs a Microchip PIC microcontroller as its central processing unit. Unfortunately, these devices are not in-circuit programmable. As a result, reprogramming the robot is a laborious process in which the microcontroller chip must be extracted, erased, re-programmed and re-inserted.

As a direct consequence of these limitations, Piglet was quickly superseded by a new design: Tigger.

4.1.3 Tigger

Tigger, which is also named after a character from *Winnie-the-Pooh*, represents a compromise between the two extremes represented by J Edgar (with a quite a lot of on-board processing power, much of it unused) and Piglet (with practically no on-board processing power). Tigger has neither a frame-grabber nor a video transmitter; instead, it has a digital camera that can be directly interfaced with the robot's CPU. This has proved to be a key element in the success of this design, since it entirely eliminates the need for a bulky, expensive and power hungry frame grabber. Tigger's specifications are given in Table 4.3. The physical configuration is shown in Figure 4.4.

Tigger has proved to be an extremely versatile design. Since its construction, it has been cloned a number of times for use in related research. To date, Tigger clones have been used in the following projects:

- Model-based visual docking [22].
- Predator-prey models using Fuzzy Cognitive Maps [156].
- Interactive voice-controlled robotics [83, 84].

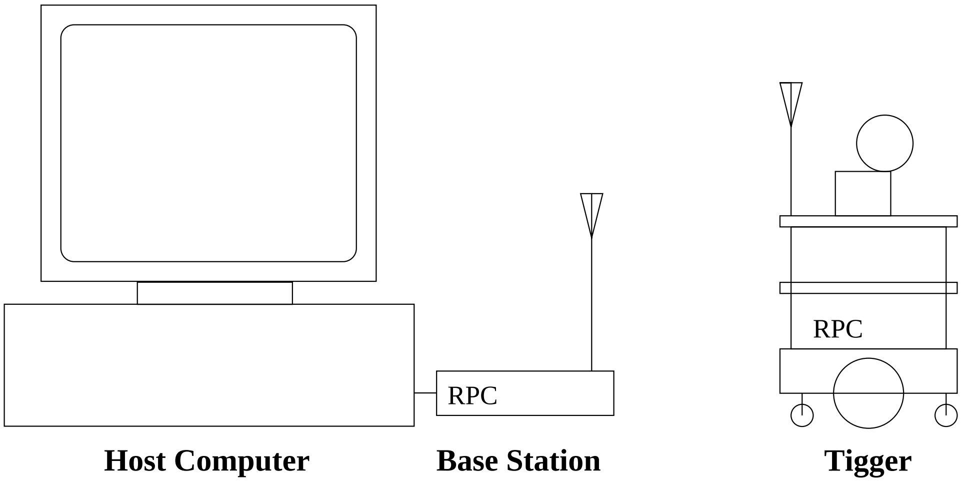


Figure 4.4: Physical setup for Tigger. Processing is divided between the host computer and the robot's on-board CPU. Communication is via a radio packet controller. Images from the digital camera are processed on-board and sent back to the host computer.

CPU	Dual microcontroller (Atmel AVR 8-bit RISC)
Sonar	None
Vision	Colour digital camera (Connectix Color Quickcam)
Comms	Radio packet controller (max throughput 40kb/s)
Programming	Via serial port
Kinematics	Twin drive wheel (non-holonomic)
Dimensions	17 $\times$ 30 cm (w $\times$ h)
Weight	1.2kg
Battery life	1 hr

Table 4.3: Specifications for robot Tigger.

- A mobile robot on the World Wide Web [166].

All of the experiments described in this thesis where conducted using either J Edgar or Tigger.

4.1.4 Kinematics

All three robots – J Edgar, Piglet and Tigger – have identical kinematics: they have a pair of independent drive wheels and a pair of passive castors (or teflon sliders). The general configuration is shown in Figure 4.5. The drive wheels are equipped with optical encoders that are used for both controlling the speed of the wheels (via a simple PID control loop), and for odometry (i.e. estimating the robot's position and orientation based on the distance each wheel has travelled). With this geometry, the robots are capable of following curves of any radius, from zero (i.e. turning in place) to infinity (i.e. moving in a straight line). The fact that the robots can turn in place, coupled with the fact that they are all roughly cylindrical, greatly simplifies obstacle avoidance problems.

Controlling the robots is a fairly straight-forward matter. Consider Figure 4.5, which shows the relationship between the individual wheel speeds (s_1, s_2), the robot's linear and angular speeds (v, ω). From the diagram, it is clear that:

$$\begin{aligned} v &= \frac{s_1 + s_2}{2} \\ \omega &= \frac{s_2 - s_1}{W}. \end{aligned} \tag{4.1}$$

Or conversely, that

$$\begin{aligned} s_1 &= v - \frac{\omega}{2W} \\ s_2 &= v + \frac{\omega}{2W}, \end{aligned} \tag{4.2}$$

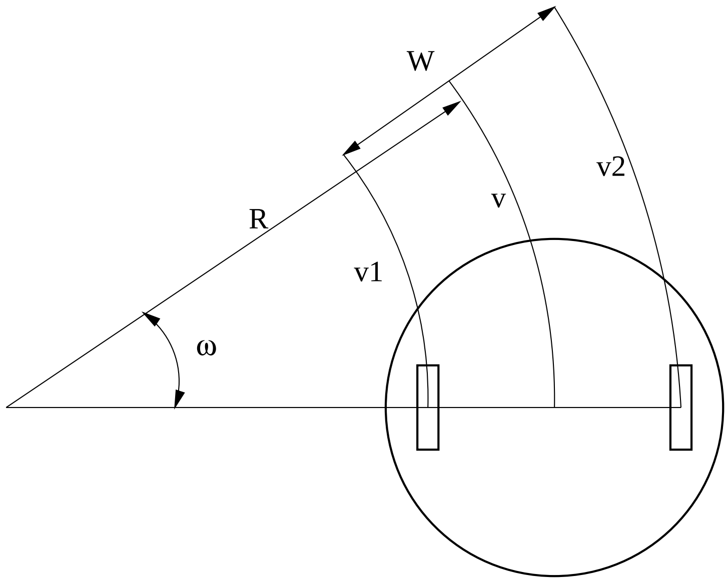


Figure 4.5: Robot kinematics. By varying the speed of the two drive wheels, the robot can be made to execute curves of any radius, from turning in place to straight-line motion.

where W is the distance between the robot's wheels. Thus, a robot can be controlled in two ways: one can either specify the target wheel speeds $(\hat{s}_1, \hat{s}_2)$, or one can specify the target linear and angular speeds $(\hat{v}, \hat{\omega})$. Naturally, since the PID control loops driving the individual motors are far from perfect, the actual speeds may differ from the target speeds.

Determining the position and orientation of a robot using odometry is also straight-forward: one simply integrates the measured velocity of the robot over time. The position (x, y) and orientation θ of the robot is therefore given by:

$$\begin{aligned} x &= \int_0^t v \cos\theta dt' \\ y &= \int_0^t v \sin\theta dt' \\ \theta &= \int_0^t \omega dt' \end{aligned} \tag{4.3}$$

This result assumes that the robot is travelling over a planar surface.

Odometry is, of course, subject to two important limitations:

- The position of the robot is measured relative to some arbitrary origin.
- The estimate will tend to 'drift' over time as errors accumulate. Sources of error include the limited resolution in the optical encoders and variation in the elevation of the surface over which the robot is travelling.

Nevertheless, odometry does provide useful information. Throughout this thesis, I will refer to the robot's *odometric position*. In doing so, I am referring to both the position (x, y) and orientation θ of the robot computed using Equation 4.3. I will also make use of the concept of an *odometric coordinate system*, which is the coordinate system in which this position is defined.

4.2 Software

In the course of this research, the basic software architecture has evolved many times. Certain elements, however, have remained constant.

- The system is object-oriented, with each module being represented by a specific class. This leads to a highly modular design in which subsystems can be developed and tested independently. It also enhances code portability, since any operating system specific code can be encapsulated within a few base classes.

- The system is multi-threaded. Each module has its own thread of execution, which facilitates multi-tasking between modules; from the programmers perspective, modules appear to run in parallel. This is an extremely useful feature; it allows, for example, the visual mapping module to be building maps at the same time as the global planning module is thinking about what the robot should be doing next.

- The system employs a message-passing mechanism. Modules communicate with one another by sending and receiving messages; all connections between modules are determined at run time and are invisible to the programmer. Furthermore, there is no central message-passing module to cause a communications bottleneck. This greatly enhances both the modularity and the *re-configurability* of the system. One can, for example, add new modules (or replace existing modules) without any modification of the existing code.

Another feature of the architecture that has not changed is the implementation language: C++. A number of other languages where considered – including Lisp and

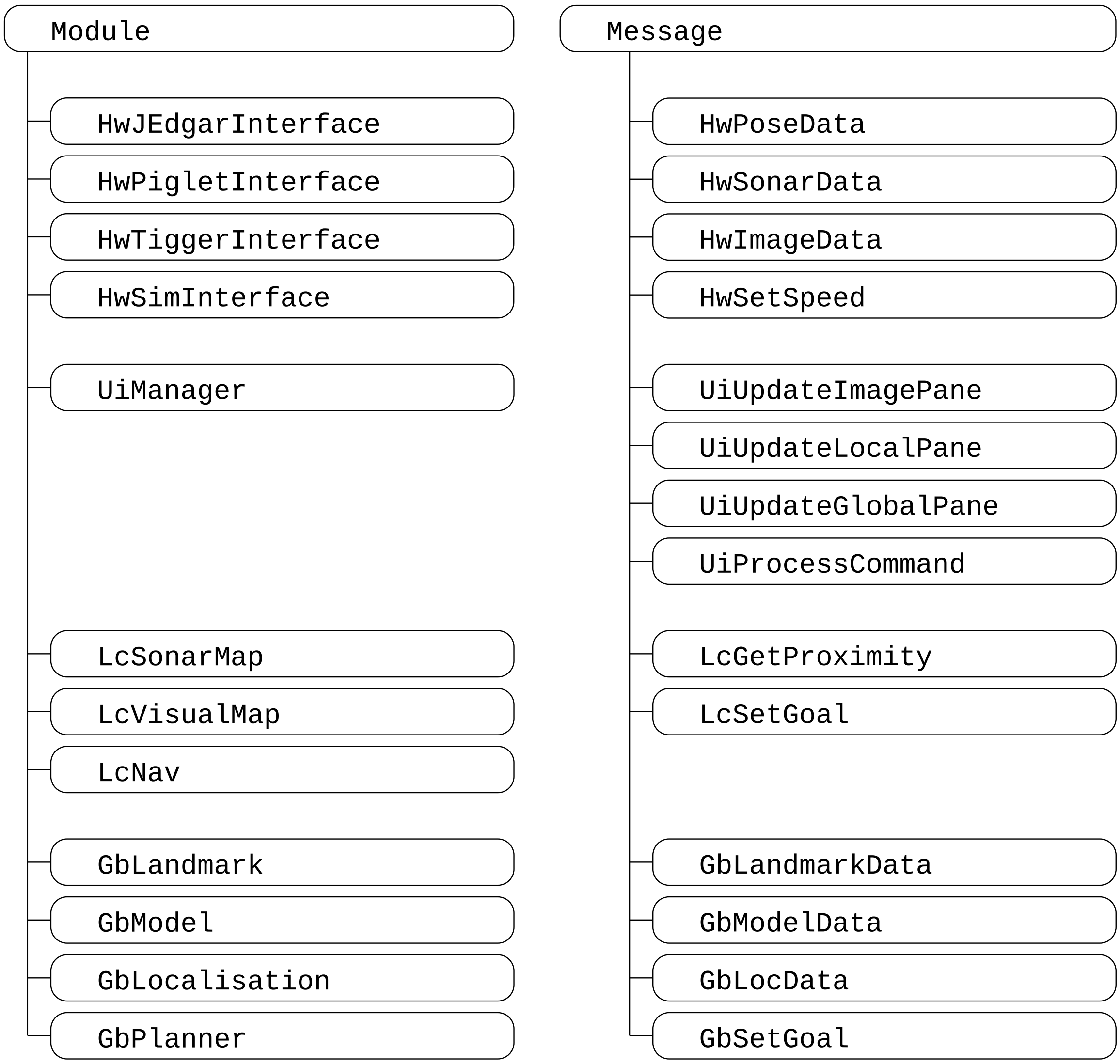


Figure 4.6: Class hierarchy chart. Some classes have been omitted for the sake of clarity.

Java – but C++ ultimately won out because of its familiarity, (relative) portability and speed.

Figure 4.6 shows a class hierarchy for the overall system. I will note just three features of this hierarchy.

- All modules are derived from the common base class `Module`. This class encapsulates a range of commonly used functions and provides support for multi-threading and message passing. A new module, with its own thread of execution and the ability to send and receive messages, can be created by simply deriving a new class from this base class.

- All messages are derived from the common base class `Message`. New messages can be created by declaring a new class that is derived from this base class. The actual mechanism used to pass messages between subsystems is entirely hidden by this class. At various times, a number of different mechanisms have been employed, including shared memory queues, UDP sockets and function lookup tables.

- Since the vast majority of the operating specific code is encapsulated within just three classes – `Module`, `Message` and `UiManager` (which is responsible for managing the user interface) – it is fairly easy to port the system to new operating systems. The original system, written under OS/2, has since been ported to Windows NT and Linux with relatively little effort.

The overall software architecture has shown itself to be extremely flexible. In addition to the research described here, this architecture has been used to support a wide range of undergraduate projects. These projects have considered topics ranging from the control of mobile robots using natural language [237] to biologically inspired models of robot interaction [156].

Chapter 5

Building Local Maps Using Sonar

> "There's a South Pole," said Christopher Robin,"and I expect there's an East Pole and a West Pole, though people don't like talking about them."

from *Winnie-The-Pooh*, by A. A. Milne.

Abstract

This chapter addresses the problem of building environment maps from sonar range data. It describes a novel mapping technique known as the response grid*, which offers a partial solution to the problem of specular reflections. The response grid can be viewed as a generalisation of the occupancy grid approach proposed by Elfes [74, 76, 77]. This chapter presents the mathematical formalism that underpins the response grid approach, and describes how this approach is implemented through the sonar mapping module.*

5.1 Introduction

Building maps from sensory data is an important aspect of mobile robot navigation. Ultrasonic range sensors are, superficially, an attractive sensor modality to use in building such maps, due mainly to their low cost, high speed and relatively simple output. Unfortunately, these sensors have a number of properties that make map building a non-trivial process.

Properties of Ultrasonic Range Finders

Standard ultrasonic range sensors, such as the widely used Polaroid device, emit a series of short ultrasonic pulses in a beam that is about 20° wide [126]. The pulses propagate through the air, reflect off solid objects and arrive back at the sensor. By timing the interval between the emitted pulse and the returned echo, these devices can measure ranges of up to about 10 meters with an accuracy of a few centimetres.

The difficulty with these devices is two-fold.

- **Limited angular resolution**. The broad beam produced by the sensor results in very poor angular resolution; i.e. the location of the object generating the reflection is known accurately in only one dimension – along the beam axis.

- **Specular reflection**. Normally, when the ultrasonic pulse encounters a surface, it undergoes *diffuse* reflection; that is, the incoming pulse is re-radiated in all directions. Therefore, regardless of the orientation of the surface, part of the pulse will be echoed back to the sensor. With some surfaces, however, the pulse will undergo *specular* reflection; that is, the pulse will reflect off the surface in the same way that light reflects off a mirror. In such cases, the pulse may not be reflected back to the sensor; in effect, the surface will appear to be invisible. Alternatively, the pulse may undergo *multiple* reflections before returning to the sensor, generating a misleading range measurement.

Environments such as offices and factories typically contain a mixture of surface types, giving rise to both diffuse and specular reflections.

The Occupancy Grid Approach

The problem of limited angular resolution can be partially overcome by combining range measurements from multiple viewpoints. Elfes [77] and Moravec [172] describe a technique in which range measurements from multiple viewpoints are combined in a two-dimensional 'occupancy grid'. With this technique, the environment is modelled as a two dimensional grid in which each cell must be in one of two states:

occupied or unoccupied. The occupancy state of each cell is determined using a simple Bayesian technique, in which measurements from multiple viewpoints are combined to determine the *probability* that the cell is occupied.

The occupancy grid has two key advantages: it is fast and it is simple. Unfortunately, it also has a major drawback – it does not cope well with specular reflections. In the occupancy grid approach, specular reflections present a contradiction – a cell may appear to be occupied when viewed from one direction, but unoccupied when viewed from another.

The Response Grid Approach

To address the problem of specular reflections (at least in part), I have developed a novel technique – called the *response grid* – that generalises the occupancy grid concept. The response grid attempts to model the behaviour of ultrasonic range sensors in a more physically realistic fashion, by accepting that a cell may generate a *response* (i.e. appear to be occupied) when viewed in one direction, but will not generate a response when viewed from another. The response grid approach has most of the advantages of the occupancy grid approach – such as speed and simplicity – but unlike the occupancy grid approach, it handles specular reflections comparatively well.

In this chapter, I will present two different methods for generating response grids from ultrasonic range data. The difference between the two methods lies in the data fusion technique they employ: the first method employs Bayesian probabilistic reasoning [28]; the second employs a Dempster-Shafer evidential reasoning [191, 218]. Both methods can be implemented in real time with modest computational resources, which makes them well suited to the task of map building.

The Sonar Mapping Module

The response grid approach is implemented through the *sonar mapping module*. This module is responsible for building a *local* map, i.e. a map describing the area within a few meters of the robot. No attempt is made to generate a *global* map, which is a much more difficult task. In later chapters, I will show how the local map can be used for both obstacle avoidance and landmark detection

Chapter Structure

The remainder of this chapter is structured as follows. Section 5.2 reviews some related approaches to sonar-based mapping. Section 5.3 develops the mathematical formalism that underpins the response grid approach, while Section 5.4 describes, in detail, the implementation of this approach through the sonar mapping module. Section 5.5 presents some experimental results obtained with the robot J Edgar. The chapter concludes with a discussion of the strengths and weaknesses of the response grid approach in Section 5.6.

Note that an early version of the work described in this chapter can be found in [119].

5.2 Related Work

There is a vast amount of literature relating either directly or indirectly to the problem of building maps from ultrasonic range data. Consequently, this review will be restricted to a few representative examples. The approaches described in these examples can be loosely grouped into three categories: grid-based approaches, feature-based approaches, and advanced sensors (i.e. ultrasonic sensors that do more than simply determine the range of an object).

Grid-based Approaches

Perhaps the most common approach to sonar mapping in use today is the *occpancy-grid* approach described by Elfes [74, 76, 77] and Moravec [172]. As stated in the Introduction, this approach models the world in terms of a regular grid in which cells may be in one of two states: *occupied* and *unoccupied*. A Bayesian data-fusion technique is applied to determine the occupancy state of each cell, based on the measurements made by an ultrasonic range sensor. Odometry is used to combine measurements taken from different viewpoints. While the occupancy grid approach is both fast and simple, it does not handle the problem of specular reflection. Pagac *at al* [189, 190] describe a straight-forward modification of the occupancy-grid approach, in which the Bayesian data-fusion technique is replaced with a technique based on Dempster-Shafer evidential reasoning [191, 218].

It is worth noting that the response grid approach described in this chapter can be thought of as a *generalisation* of both the original occupancy grid approach (due to Elfes and Moravec) and its Dempster-Shafer variant (due to Pagac *et al*). As I will show in Section 5.3, one can recover either of these formulations by choosing an appropriate set of parameters for the response grid.

Beckerman *et al* [24] take an interesting alternative approach to the problem of building grid-based maps with sonar. Whereas the occupancy grid approach is principally concerned with managing *stochastic* errors (such as errors due to noise in the sensor), the approach described by Beckerman is principally concerned with *systematic* errors. In particular, the authors are concerned with the errors that arise from the limited angular resolution of standard ultrasonic range sensors. Their solution entails a four-valued labelling scheme in which each cell in the grid is assigned one of four labels – *occupied*, *empty*, *unknown*, or *conflict* (meaning there is disagreement about the state of a cell). Simple logic is used to combine cell labellings arising from different range measurements. This logic is, however, driven by an understanding of the underlying physical properties of the sensor. Up to this point, the approach described by these authors is very much like a discrete logic version of the original occupancy grid approach. However, the authors introduce one more step: a *labelling consistency algorithm*. This algorithm is applied to force some cells to change state. For example, isolated 'occupied' cells are changed to 'unknown' cells, on the assumption that there are no point-like objects in the environment. This is very different from the occupancy grid approach, in which each cell is treated in isolation. With this approach, the state of a cell depends, at least in part, upon the state of its neighbours. Potentially, this could be a very useful property. Unfortunately, despite some musings on the subject by the authors, it is not clear how this approach could be extended to handle specular reflections.

Harris [108] describes a biologically inspired approach in which the occupancy grid is replaced with a network of artificial neurons. The neurons have lateral connections that allow likely features (such as lines) to be reinforced, and unlikely features (such as random scatter) to be inhibited. While this approach is potentially interesting, the work described in [108] is very preliminary. It does not, for example, consider the issue of how the network should be trained to maximise the quality of the map (the network, as described, uses hand-coded weights). In principle, however, this approach *might* be able to solve the problem of specular reflections.

Thrun [243] also uses a neural network for building occupancy maps. Readings from a number of sensors are fed into a neural network, which then determines the probability that a cell is occupied. The network is trained, oddly enough, on simulation data. Nevertheless, very good results are reported. The key advantage of this approach is that readings from multiple sensors can be processed at the same time; in some cases, this allows specular reflections to be suppressed, as the network considers the readings obtained from two or more different range sensors. Note that, this approach is quite different from that proposed by Harris: there are no lateral connections between cells, so each cell is treated independently.

Feature-based Approaches

One of the earlier attempts at handling the problems associated with specular reflections is due to Brown [43]. He writes:

> Using an ultrasonic sensor to look at arbitrary objects in a room is rather like standing in a room completely filled with mirrored objects and having only a penlight glued to your forehead as a source of light...

Brown considers the problem of an ultrasonic range sensor mounted on a robot arm. He develops a mathematical theory for determining the bearing of a planer surface (which gives rise to specular reflections), based on measurements taken from three different positions (or by three well spaced sensors). He goes on to show how the robot might 'track' that surface, much in the way that a mobile robot might follow a wall. Unfortunately, despite the elegance of this approach, it is unclear how well it would 'scale up' to a mobile robot applications, where the environment is much more cluttered, and contains a mixture of surface types (both diffuse and specular).

Bozma *et al* [33] also look at the problem of specular environments. They go further than Brown, developing a method for classifying environmental features into planes, corners and edges (or outside-corners). Planes and corners are distinguished from edges by *rotating* the sensor; planes are distinguished from corners by *translating* the sensor. These features are then used to build an environment map. The limitations of this approach are two-fold: it works only in environments made up of specular surfaces (diffuse surfaces are not handled), and features that are close together cannot be resolved (due to the limited angular resolution of the sensor).

In a related piece of work, Kuc *et al* [150] describe a navigation strategy that makes use of the approach described by Bozma. This strategy is designed to ensure that the robot builds an accurate and complete map as it goes about reaching its goal. Using its knowledge of the physical properties of the sensor, the robot can follow a path that ensures that it does not have any nasty surprises, such as colliding with a wall it has not detected. Unfortunately, as is the case with [33], this approach works only in environments made entirely of specular surfaces.

Lee [155] builds on the earlier work on Leonard *et al* [158] to construct feature-based maps of the environment. With this approach, similar sonar readings are grouped to form *regions of constant depth* (RCD's). This reduces the uncertainty associated with bearing of a detected object, since the object must lie in the overlap between the various sonar beams making up the RCD. Individual RCD's are then formed into clusters, with each cluster corresponding to a possible feature, such as a plane or an edge. When a cluster becomes big enough, it is added to the map. The advantage of this approach is that it builds very nice maps, even in environments containing a mixture of diffuse and specular surfaces. Unfortunately, it is also prone to missing certain features, since there may not be enough range readings to form either an RCD or a significant cluster. In addition, this method is quite slow.

The final feature-based approach I will consider is due to Kurz [152] (in this case, 'feature-based' is used in the loosest possible sense). Kurz describes a method for building a *topological* map of the environment, in which each node corresponds to a particular sensor 'situation'. These 'situations' are *learnt automatically* by the robot by applying an unsupervised clustering algorithm to the data obtained from a ring of ultrasonic range sensors. While the results are good, this approach is primarily focused on the problem of building *global* maps for localisation, rather than the problem of building *local* maps for obstacle avoidance. The 'situations', in particular, are difficult to interpret, since they do not necessarily correspond to the kind of 'situations' that would be understood by a human.

Advanced Sensors

A number of authors have looked at the sonar mapping problem from a hardware perspective, and have designed sensors that are much more sophisticated than the simple time-of-flight range sensors that are normally used. A general comment that could be applied to most of these sensors is: 'nice if you've got them'.

Ohya *et al* [183] describe a special sensor that is capable of measuring the *orientation* of a surface. The sensor has one transmitter and two receivers; the orientation of a reflecting surface can be determined from phase difference measured by these receivers. Thus, the sensor returns two pieces of information: the range to a surface, and the orientation of that surface with respect to the sensor. Armed with this information, one can build a *vector* representation of the environment, in which each vector represents a surface normal. Some basic continuity conditions are then applied to form these surface normals into features, such as walls and corners. The map building process is significantly simplified by the availability of surface orientation information.

Bozma *et al* [34, 35] also describe a special sensor, but one which requires only a single transducer. By measuring both the echo-energy and echo-duration of the return signal (as opposed to just the time-of-flight), it is possible to determine the *texture* of the reflecting surface. That is, the surface can be classified as smooth, rough, very rough, etc. A feature-based map is then constructed, in which the surface texture is used to distinguish between types of feature.

Yata *et al* [261] present a method for accurately determining both the range and bearing of an object with a single ultrasonic transducer. Since the transducer is not equally sensitive in all directions, the power spectrum of the return signal will depend upon the bearing of the reflecting object. Hence, by carefully monitoring this spectrum, it is possible to determine both the bearing and range of the object with a reasonable degree of accuracy (there is, unfortunately, an ambiguity in the *sign* of the bearing, due to the symmetric nature of the transducer). As it stands, this method is quite computationally expensive. While the authors suggest that the computation could be done in hardware, at this point, such hardware remains a proposal only.

Kleeman *et al* [137] describe a quite remarkable ultrasonic sensor that is capable of determining both the range and bearing of objects to a very high degree of accuracy (sub-centimetre in range, and sub-degree in bearing). The sensor is also able to classify objects into planes, corners and edges. Physically, the sensor is made up of two transmitters and two receivers, spaced about 20cm apart (actually, the sensor has three *transducers* – one transmitter, one receiver and one transmitter/receiver). The sensor is able to measure the bearing of objects by considering the phase difference between the signals measured by the two receivers. It is able to distinguish between different types of objects by using the two transmitters. Perhaps the only drawback with this type of sensor is its size, and the amount of computation required to generate the final measurements. A three dimensional version of this sensor, which can measure the range, bearing and *elevation* of an object is described Akbarally [3].

Unfortunately, advanced sensors of this kind are not readily available. Therefore, the response grid approach described in this chapter makes use of standard ultrasonic range sensors. These are commercially available, off-the-shelf devices.

5.3 Mathematical Formalism

5.3.1 The Response Grid

The response grid models the environment as a set of cells arranged in a regular two-dimensional grid. A given cell in the grid may either be entirely empty, or else it may contain one or more surfaces which reflect ultrasonic pulses (see, for example, Figure 5.1). A pulse entering a cell will do one of three things:

- If the cell is entirely empty, the pulse will pass through the cell unaffected.
- If the cell contains one or more reflective surfaces, the pulse may be reflected *back* to the ultrasonic detector.
- If the cell contains one or more reflective surfaces, the pulse may be reflected *away* from the ultrasonic detector.

The behaviour of a pulse reflecting off a surface contained in the cell will depend upon the orientation of the surface and the direction of the incoming pulse. If a pulse which is propagating in some given direction is reflected back to the detector, the cell is said to have a *response* in that direction. Furthermore, any cell having a response in one or more directions is assumed to be *occupied*.

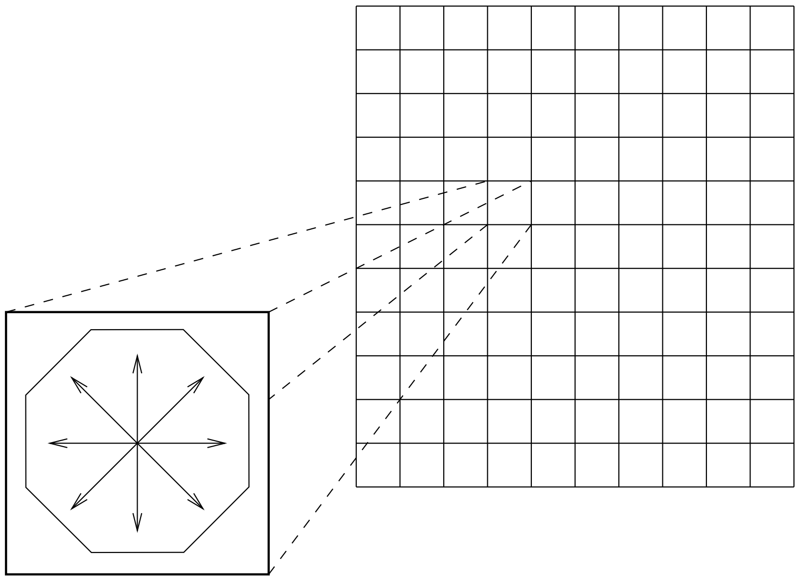


Figure 5.1: Illustration of the response grid approach. In this particular grid, each cell may have up to 8 response directions ($n = 8$).

Note that this leaves open the possibility of a cell being occupied, but not having a response in *any* direction. This situation can arise either when the cell contains a material that entirely absorbs incident pulses, or else contains a surface that reflects the pulses in such a way that they can *never* return to the detector. The former case arises rarely in real world situations, and can be ignored; the latter case requires more careful consideration. The model that is proposed here is two dimensional, being intended for use with mobile robots whose ultrasonic sensors are constrained to motion in a plane. To an ultrasonic sensor so constrained, certain three-dimensional shapes are effectively invisible. Thus any algorithm using ultrasonic sensor data to determine occupancy must fail under certain environmental conditions. Fortunately, such conditions are rare in most environments, particularly human-made environments such as offices and factories.

A second problem arises when a surface reflects pulses in such a way that they travel to another surface and are then reflected back to the detector. Without some way of knowing that this 'multiple reflection' effect has occurred, the detected response will be attributed to the wrong cell. Consequently, a cell which is in fact empty may appear to be occupied. The method described in this chapter does not explicitly model such effects; as a result, multiple reflections can manifest themselves as artefacts in the generated maps.

The model described above can be expressed mathematically as follows. The *occupancy* of a cell at location (x, y) is measured by the state variable $\mathrm{Occ}(x, y)$ which can have one of two values:

$$\mathrm{Occ}(x, y) = [\mathit{occupied}, \mathit{unoccupied}]. \tag{5.1}$$

The *response* of a cell in some direction i is measured by the state variable $\mathrm{Res}(x, y, i)$:

$$\mathrm{Res}(x, y, i) = [\textit{response}, \textit{no response}]. \tag{5.2}$$

The direction i is allowed to take integer values between 1 and n. The two state variables are bound together by a logical implication. In order to express this clearly, we define a set of propositions:

$$\begin{aligned} O_{x,y} &: \mathrm{Occ}(x, y) = \textit{occupied} \\ R_{x,y,i} &: \mathrm{Res}(x, y, i) = \textit{response}. \end{aligned} \tag{5.3}$$

That is, the proposition $O_{x,y}$ states that the cell at (x, y) is occupied, and the proposition $R_{x,y,i}$ states that the cell at (x, y) generates a response in direction i. We can therefore write the implication:

$$O_{x,y} \Leftarrow R_{x,y,1} \vee R_{x,y,2} \cdots \vee R_{x,y,n-1} \vee R_{x,y,n}. \tag{5.4}$$

That is, the cell is said to be occupied if it generates a response in any of n different directions; and unoccupied if *fails* to generate a response in all n directions. Note that while the response grid is very good at determining which cells are occupied; it is not very good at determining which cells are *unoccupied*. A cell can only be said to be occupied if it has been viewed from every possible direction. Unfortunately, this does limit the applicability of this technique to robot navigation tasks – it tends to lead to very conservative behaviour on the part of the robot.

In the following sections, I will present two different methods for using ultrasonic range data to determine the response properties of each cell, and thence to determine its occupancy. The first method employs Bayesian probabilistic reasoning [28], the second employs Dempster-Shafer evidential reasoning [191, 218].

5.3.2 Method 1: Bayesian

The objective of the Bayesian method is to determine, for each cell, the *probability* that the cell is occupied. That is, for a cell at (x, y) we wish to determine the probability that the proposition $O_{x,y}$ is true. Rather than attempting to calculate this probability directly from the range data, we compute it indirectly. Since $O_{x,y}$ is related to the cell responses by the logical implication expressed in Equation 5.4, the probability that the cell is occupied must be given by:

$$p(O_{x,y}) = p(R_{x,y,1} \vee R_{x,y,2} \cdots \vee R_{x,y,n-1} \vee R_{x,y,n}). \tag{5.5}$$

To expand the right hand side of this equation, we make use of the fact that for two independent propositions A and B, the probability of either A or B being true is given by [28]

$$p(A \vee B) = p(A) + p(B) - p(A)p(B). \tag{5.6}$$

Applying this to the above equation, one obtains after a little algebra:

$$p(O_{x,y}) = 1 - \Pi_i(1 - p(R_{x,y,i})). \tag{5.7}$$

This expression can be used to compute the probability that a cell is occupied once we have determined the cell response probabilities. Note that this involves the determination of n *separate* probabilities for each cell, since $\{R_{x,y,1} \cdots R_{x,y,n}\}$ is a set of n *independent* propositions.

Consider now just one proposition $R_{x,y,i}$ corresponding to the response in direction i for a cell at (x, y). We can apply Bayes' rule to determine the probability that $R_{x,y,i}$ is true, given that the sonar sensor has measured an object with range r and bearing θ:

$$p(R_{x,y,i} \mid r, \theta) = \frac{p(r, \theta \mid R_{x,y,i})}{p(r, \theta)} p(R_{x,y,i}). \tag{5.8}$$

In this expression $p(R_{x,y,i})$ is the *prior probability* of obtaining a response. In Bayesian approaches it is very common to set the prior probability to 0.5 to indicate no opinion. However, in our case this leads to undesirable results. Consider a single cell for which we have no measurements; applying Equation 5.7 to determine the probability that the cell is occupied would yield $p(O_{x,y}) = 1 - 0.5^n$. So for $n = 8$ we would have $p(O_{x,y}) = 0.996$. That is, on the basis of no measurements whatsoever we would conclude with near certainty that the cell is occupied. A better way to set the prior probabilities in this case is to demand that $p(O_{x,y}) = 0.5$ in the absence of any measurements. This implies that we should set the prior probability to:

$$p(R_{x,y,i}) = 1 - (0.5)^{\frac{1}{n}}. \tag{5.9}$$

where n is the number of response directions.

The denominator in Equation 5.8 gives the probability of obtaining the measurement (r, θ), and can be thought of as a normalisation constant. That is, $p(r, \theta)$ must be such that:

$$p(R_{x,y,i} \mid r, \theta) + p(\neg R_{x,y,i} \mid r, \theta) = 1 \tag{5.10}$$

where $\neg R_{x,y,i}$ denotes the proposition that the cell at (x, y) *does not* produce a response in direction i. The probability $p(\neg R_{x,y,i} \mid r, \theta)$ is also computed using Bayes' Law:

$$p(\neg R_{x,y,i} \mid r, \theta) = \frac{p(r, \theta \mid \neg R_{x,y,i})}{p(r, \theta)} p(\neg R_{x,y,i}). \tag{5.11}$$

The most important term in Equation 5.8 and Equation 5.11 is the *sensor model* described by $p(r, \theta \mid R_{x,y,i})$ and $p(r, \theta \mid \neg R_{x,y,i})$. The first term indicates the probability of obtaining the measurement (r, θ), given that the proposition $R_{x,y,i}$ is true; while the second term indicates the probability of obtaining the measurement (r, θ), given that the proposition $R_{x,y,i}$ is false. The sensor model is the key element in the Bayesian formalism, since it encodes all of the robot's knowledge about the properties of the sensor. Qualitatively, the sensor model used by the sonar mapping module can be expressed as follows:

- If the cell's bearing is such that it lies within the detection region of the sensor (see Figure 5.2), *and* the bearing of the cell is the same as the response direction i, *and* the cell is closer to the sensor than the recorded range, the cell probably does not a generate response in this direction:

$$\begin{aligned} p(r, \theta \mid R_{x,y,i}) &= 0.1 \\ p(r, \theta \mid \neg R_{x,y,i}) &= 0.9. \end{aligned} \tag{5.12}$$

- If the cell's bearing is such that it lies within the detection region of the sensor, *and* the bearing of the cell is the same as the response direction i, *and* the cell is at about the same distance from the sensor as the recorded range, the cell *may* generate a response in this direction.

$$\begin{aligned} p(r, \theta \mid R_{x,y,i}) &= \alpha/r \\ p(r, \theta \mid \neg R_{x,y,i}) &= 0.1. \end{aligned} \tag{5.13}$$

- Otherwise, we know nothing about this cell:

$$\begin{aligned} p(r, \theta \mid R_{x,y,i}) &= 0.5 \\ p(r, \theta \mid \neg R_{x,y,i}) &= 0.5. \end{aligned} \tag{5.14}$$

The first and last parts of this model are self-explanatory, but the second requires some explanation. When the sensor records a range r we know that *something* at this range has generated a response. Due to the finite angular resolution of the detector, however, there may be a number of cells that may have generated this response. In general, the number of such cells will be proportional to the measured range, so the probability that any individual cell generated the response is *inversely proportional* to the measured range. In effect, we give more weight to short range measurements than to long range measurements. This rule is both physically plausible and intuitively appealing. The α that appears in the above rule is a normalisation constant: summing over the probabilities assigned to each cell at range r should yield a total probability of 1.

Note that, in practice, we only need to update those cells that lie within the detection region of the sensor. The appropriate response probabilities will be either increased or decreased, according to the sensor model given above.

While it is possible to construct more complex sensor models than the one given above, this heuristic model is sufficient for most purposes.[1] Note also that the original occupancy grid approach (due to Elfes [74, 76, 77]) can be recovered (albeit with a somewhat simplified sensor model) by setting the number of response directions n equal to 1.

Equations 5.7 and 5.8 provide us with the mathematical tools required to construct a Bayesian response grid. A discussion of the practical implementation of such a grid is presented in Section 5.4, where the implementation of both the Bayesian and Dempster-Shafer methods is considered.

[1] Constructing a sensor model that is both mathematically and physically justifiable is a somewhat involved process. Since this process makes use of Kolmogoroff's theorem (which involves a summation over all possible world states), one must have a detailed understanding not only of the sensor, but of the environment in which it is to be used. See [75] for details.

5.3.3 Method 2: Dempster-Shafer

The Dempster-Shafer Theory of Evidence [142, 191, 218] can be applied to the response grid in a straight-forward fashion. The objective is to determine the *support* for the proposition $O_{x,y}$; that is, the amount of evidence for the proposition that the cell at (x, y) is occupied. From the logical implication expressed in Equation 5.4, we can write the following:

$$\mathrm{Sup}(O_{x,y}) = \mathrm{Sup}(R_{x,y,1} \vee R_{x,y,2} \cdots \vee R_{x,y,n-1} \vee R_{x,y,n}). \tag{5.15}$$

To expand the right hand side of this equation, the Dempster-Shafer analogue of Equation 5.7 is used. Given two independent propositions A and B, the support for the combined proposition $A \vee B$ is given by [16]:

$$\mathrm{Sup}(A \vee B) = \mathrm{Sup}(A) + \mathrm{Sup}(B) - \mathrm{Sup}(A)\mathrm{Sup}(B). \tag{5.16}$$

Applying this to Equation 5.15, one obtains:

$$\mathrm{Sup}(O_{x,y}) = 1 - \Pi_i(1 - \mathrm{Sup}(R_{x,y,i})), \tag{5.17}$$

which allows us to calculate the support for the proposition that a cell is occupied, once we have determined the support for each of the propositions $\{R_{x,y,1} \cdots R_{x,y,n}\}$.

Consider now just one proposition $R_{x,y,i}$, corresponding to the response in direction i for a cell at (x, y). In order to determine the support for the proposition that the cell responds in this direction, the Dempster-Shafer approach requires that we construct a relevant *frame of discernment*, which is a set containing all the propositions of interest. In our case, the frame of discernment contains just two propositions: that the cell responds, or that it does not:

$$\Theta = \{R_{x,y,i}, \neg R_{x,y,i}\}. \tag{5.18}$$

In general, support for these propositions cannot be determined directly from the available evidence. Instead, support is computed indirectly via a *mass distribution*. The mass distribution allows us to allocate a 'weight' to any element in the frame of discernment, or to any proper *subset* of the frame of discernment. Thus, in this application, the mass distribution can assign weight to any element in the set:

$$2^{\Theta} = \{R_{x,y,i}, \neg R_{x,y,i}, R_{x,y,i} \vee \neg R_{x,y,i}\}. \tag{5.19}$$

The weight assigned to an element in this set indicates the amount of evidence that supports that element directly. Support for propositions in the frame of discernment can be calculated according to the rule:

$$\mathrm{Sup}(A) = \sum_{B \subseteq A} m(B). \tag{5.20}$$

Thus the support for the proposition $R_{x,y,i}$ is trivially equal to

$$\mathrm{Sup}(R_{x,y,i}) = m(R_{x,y,i}). \tag{5.21}$$

The mass distribution is somewhat similar to a probability distribution in that it must be non-negative and must sum to one. Consider the following mass distribution:

$$\begin{aligned} m(R_{x,y,i}) &= 0 \\ m(\neg R_{x,y,i}) &= 0 \\ m(R_{x,y,i} \vee \neg R_{x,y,i}) &= 1. \end{aligned} \tag{5.22}$$

For brevity we write this as $m = (0, 0, 1)$. This distribution corresponds to complete ignorance, since it will yield no support for either $R_{x,y,i}$ or its negation. Contrast this with the distribution $m = (1, 0, 0)$, which indicates complete support for $R_{x,y,i}$ and no support for its negation; and with $m = (0.5, 0.5, 0)$, which indicates a contradiction – there is equal support for $R_{x,y,i}$ and its negation. One of the attractive features of the Dempster-Shafer approach is the way in which ignorance and contradiction are clearly distinguished; this is not true of the Bayesian approach.

In our problem, every new measurement arriving from the ultrasonic sensor is treated as a new piece of evidence for which we must generate a mass distribution. These separate distributions can then be combined to generate a collective opinion. If $m(A \mid r_1, \theta_1)$ and $m(B \mid r_2, \theta_2)$ represent mass distributions arising from a pair of independent measurements (r_1, θ_1) and (r_2, θ_2), the combined mass distribution $m(C \mid r_1, \theta_1; r_2, \theta_2)$ is given by Dempster's rule [160]:

$$m(C \mid r_1, \theta_1; r_2, \theta_2) = \frac{1}{1-\kappa} \sum_{A \cap B = C} m(A \mid r_1, \theta_1) m(B \mid r_2, \theta_2), \tag{5.23}$$

where κ is defined as

$$\kappa = \sum_{A \cap B = \emptyset} m(A \mid r_1, \theta_1) m(B \mid r_2, \theta_2), \tag{5.24}$$

and A, B and C are elements of the set $\{R_\phi, \neg R_\phi, R_\phi \vee \neg R_\phi\}$. Inspecting the above equation, one can see that when a contradiction exists (for example when there is support for both a proposition and its negation), κ will be non-zero. As a result κ is usually interpreted as indicating the degree of contradiction between the two mass distributions. Dempster's rule can be thought of as Dempster-Shafer equivalent of Bayes' rule.

The rule used by the sonar mapping module for generating mass distributions from range measurements is analogous to the sensor model used in the Bayesian case. Qualitatively, it is expressed as follows:

- If the cell's bearing is such that it lies within the detection region of the sensor, *and* the bearing of the cell is the same as the response direction i, *and* the cell is closer to the sensor than the recorded range, then there is evidence that the cell does not generate a response in this direction:

$$\begin{aligned} m(R_{x,y,i} \mid r,\theta) &= 0 \\ m(\neg R_{x,y,i} \mid r,\theta) &= 0.9 \\ m(R_{x,y,i} \vee \neg R_{x,y,i} \mid r,\theta) &= 0.1. \end{aligned} \tag{5.25}$$

- If the cell's bearing is such that it lies within the detection region of the sensor, *and* the bearing of the cell is the same as the response direction i, *and* the cell is the same distance from the sensor as the recorded range, then there is evidence that the cell may generate a response in this direction:

$$\begin{aligned} m(R_{x,y,i} \mid r,\theta) &= \alpha/r \\ m(\neg R_{x,y,i} \mid r,\theta) &= 0 \\ m(R_{x,y,i} \vee \neg R_{x,y,i} \mid r,\theta) &= 1 - \alpha/r. \end{aligned} \tag{5.26}$$

- Otherwise, we know nothing about the response of this cell:

$$\begin{aligned} m(R_{x,y,i} \mid r,\theta) &= 0 \\ m(\neg R_{x,y,i} \mid r,\theta) &= 0 \\ m(R_{x,y,i} \vee \neg R_{x,y,i} \mid r,\theta) &= 1. \end{aligned} \tag{5.27}$$

The reasoning used to arrive at this model is identical to that used in the Bayesian case. Note, however, that the initial, or prior, mass distribution used corresponds to complete ignorance – $m = (0, 0, 1)$.

The implementation of the Dempster-Shafer method is considered in the next section.

5.4 The Sonar Mapping Module

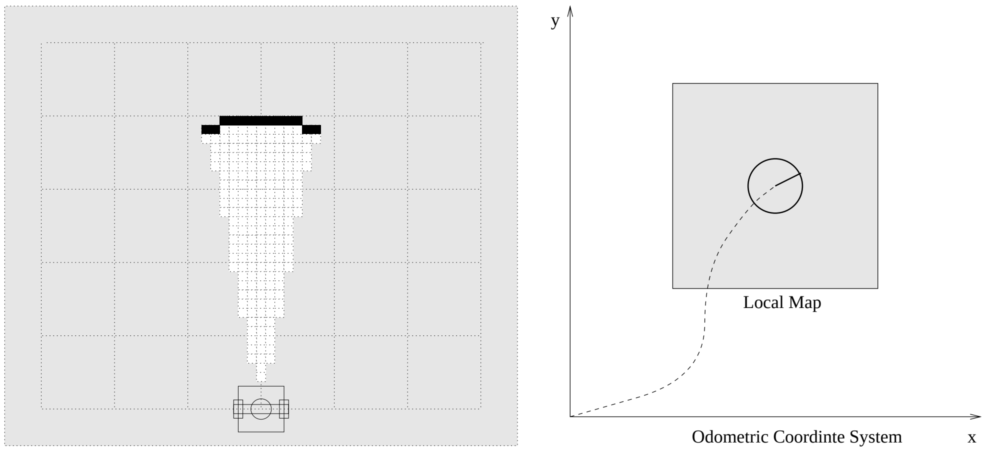
The sonar mapping module is capable of implementing both of the update methods described in the previous section. The implementation for these methods is, in fact, remarkably similar.

The map is represented by a two-dimensional array, with each cell corresponding to a small region of the environment. Associated with each cell is a value representing the cell's occupancy, together with an array of values representing the cell's responses. The meaning of these values will depend upon the particular update method being used. Odometry is used to combine measurements taken from different robot locations. Unfortunately, since odometry is prone to cumulative drift, it can only be used over short distances. Consequently, the sonar mapping module does not attempt to build a map of the robot's *global* environment. Instead, it builds a *local* map. The local map can be thought of as a moving 'window' defined with respect to the odometric coordinate system. As the robot moves, the window is also moved so that it remains centred around the robot (strictly speaking, the window is moved whenever the robot leaves the central cell in the map, and it is always moved by an integral number of cells). This situation is illustrated in Figure 5.2.

The map is updated in response to two different events.

- When the robot moves, the entire map is translated. As a result, some cells will 'fall off' the edge of the map (at which point the information encoded in these cells is lost). At the same time, new cells will be added at the opposite side of the map. These cells are initialised to some set of prior values according to the rules described in the following sections.

- When a sonar measurement is taken, the sonar mapping module first deter-



(a) (b)

Figure 5.2: (a) The sensor model. Only those cells lying between the robot and the detected obstacle in an arc of 20° are updated. (b) Relationship between the odometric coordinate system and the local map. The local map 'window' remains centered around the robot at all times.

mines which of the cells in the array *need* to be updated. Since the pulse emitted by the ultrasonic sensor propagates outward through a conical region of space, only the cells corresponding to this region of space need be updated. See, for example, Figure 5.2. Furthermore, for each of the candidate cells, only one of the response directions needs to be updated – that corresponding to the direction of the emitted pulse. Once this determination has been made, the appropriate cell response can be updated and the overall cell occupancy recomputed. The details of the update process depend upon the method being used, as described in the following sections.

Note that, for the sake of speed, many of the operations used in this process are pre-computed, with the results stored in look-up tables.

5.4.1 Method 1: Bayesian

Consider first the Bayesian method. The cell occupancy is represented by the probability $p(O_{x,y})$ and the cell responses by an array of probabilities $p(R_{x,y,i})$. These probabilities are initialised according to the rules discussed in Section 5.3.2:

$$\begin{aligned} p(R_{x,y,i}) &\leftarrow 1 - 0.5^{\frac{1}{n}} \\ p(O_{x,y}) &\leftarrow 0.5. \end{aligned} \tag{5.28}$$

This initialisation is also applied to cells that are 'added' when the map is translated. When the sonar sensor generates a measurement (r, θ), a single response probability is updated as follows:

$$p(R_{x,y,i}) \leftarrow \frac{p(r,\theta \mid R_{x,y,i})p(R_{x,y,i})}{p(r,\theta \mid R_{x,y,i})p(R_{x,y,i}) + p(r,\theta \mid \neg R_{x,y,i})p(\neg R_{x,y,i})} \tag{5.29}$$

This is simply Bayes' law (equation 5.8) recast in an incremental form, with the current response probability used as the prior probability. The sensor model, Equations 5.12 to 5.14, is used to determine $p(r, \theta \mid R_{x,y,i})$ and $p(r, \theta \mid \neg R_{x,y,i})$. There is no attempt to interpolate between response directions – the measurement is simply 'binned', and one response probability is updated. A more sophisticated version (which has not been implemented) would update adjacent response directions.

The new cell occupancy $p(O_{x,y})$ is recomputed using Equation 5.7.

$$p(O_{x,y}) \leftarrow 1 - \Pi_i(1 - p(R_{x,y,i})). \tag{5.30}$$

Note that the incremental form of Bayes' Law (Equation 5.29) assumes that successive measurements are *statistically independent* [28]; i.e. that the uncertainty associated with each measurement is due to random, not systematic, effects. Since this is most certainly *not* the case for sonar measurements (the majority of the uncertainty is associated with the finite angular resolution of the sensor), the overall validity of the response grid approach must be tested experimentally.

5.4.2 Method 2: Dempster-Shafer

In the Dempster-Shafer implementation, cell occupancy is represented by the support $\mathrm{Sup}(O_{x,y})$, and cell responses by an array of mass distributions $m(R_{x,y,i})$, $m(\neg R_{x,y,i})$, $m(R_{x,y,i} \vee \neg R_{x,y,i})$. These are initialised to 'don't know' values:

$$\begin{aligned} m(R_{x,y,i}) &\leftarrow 0 \\ m(\neg R_{x,y,i}) &\leftarrow 0 \\ m(R_{x,y,i} \vee \neg R_{x,y,i}) &\leftarrow 1 \\ \mathrm{Sup}(O_{x,y}) &\leftarrow 0. \end{aligned} \tag{5.31}$$

This initialisation is also applied to cells that are 'added' when the map is translated. When the sonar sensor generates a measurement (r, θ), the mass distribution for a single response direction is updated as follows:

$$\begin{aligned}
m(R_{x,y,i}) &\leftarrow \frac{1}{1-\kappa}[m(R_{x,y,i} \mid r,\theta)m(R_{x,y,i}) \\
&\quad + m(R_{x,y,i} \vee \neg R_{x,y,i} \mid r,\theta)m(R_{x,y,i}) \\
&\quad + m(R_{x,y,i} \mid r,\theta)m(R_{x,y,i} \vee \neg R_{x,y,i})] \\
m(\neg R_{x,y,i}) &\leftarrow \frac{1}{1-\kappa}[m(\neg R_{x,y,i} \mid r,\theta)m(\neg R_{x,y,i}) \\
&\quad + m(R_{x,y,i} \vee \neg R_{x,y,i} \mid r,\theta)m(\neg R_{x,y,i}) \\
&\quad + m(\neg R_{x,y,i} \mid r,\theta)m(R_{x,y,i} \vee \neg R_{x,y,i})] \\
m(R_{x,y,i} \vee \neg R_{x,y,i}) &\leftarrow \frac{1}{1-\kappa}[m(R_{x,y,i} \vee \neg R_{x,y,i} \mid r,\theta)m(R_{x,y,i} \vee \neg R_{x,y,i})]
\end{aligned} \tag{5.32}$$

where

$$\kappa = m(R_{x,y,i} \mid r,\theta)m(\neg R_{x,y,i}) + m(\neg R_{x,y,i} \mid r,\theta)m(R_{x,y,i}). \tag{5.33}$$

The three expressions in Equation 5.32 represent an expanded version of Dempster's rule, Equation 5.23 (the third expression is, in fact, redundant; $m(R_{x,y,i} \vee \neg R_{x,y,i})$ can be computed from the normalisation condition). The sensor model, Equations 5.25 to 5.27, is used to determine the value of the terms $m(R_{x,y,i} \mid r,\theta)$, $m(\neg R_{x,y,i} \mid r,\theta)$, $m(R_{x,y,i} \vee \neg R_{x,y,i} \mid r,\theta)$. The new cell occupancy $\mathrm{Sup}(O_{x,y})$ is recomputed using Equation 5.16:

$$\mathrm{Sup}(O_{x,y}) = 1 - \Pi_i[1 - m(R_{x,y,i})], \tag{5.34}$$

where we have made use of Equation 5.21 to rewrite the support in terms of the mass distribution.

Note that, as in the Bayesian case, the incremental form of Dempster's Rule (Equation 5.32) assumes that successive measurements are *independent* [191, 218]. This assumption must be validated experimentally.

5.5 Experimental results

The results presented in this section were obtained using the robot J Edgar, which is equipped with a single Texas Instruments/Polaroid ultrasonic range-finder attached to a pivoting head (Figure 5.3). The sensor has an unobstructed view and can rotate through 360° in 7.5° increments. Consequently, 48 range readings are generated by each 'sweep' of the pivoting head. Figure 5.3 shows the environment in which the experiments were conducted; this is a relatively complex environment containing a number of boxes, a hatstand and a chair. The boxes were deliberately chosen because they give rise to specular reflections. The robot was allowed to travel in a more-or-less straight line between the obstacles, taking range readings as it went. The results presented in this section include readings from about 30 complete sweeps of the sensor head, i.e. about 1440 individual readings. The robot's location was determined by odometry.

Figure 5.4 shows the maps produced by the two different methods (Bayesian and Dempster-Shafer) for varying values of n (i.e. varying numbers of response directions). Each pixel in these maps represents a region 4 centimetres square. Cells which are probably occupied (or for which there is strong support) appear darker than cells which are probably unoccupied. The dotted line shows the path taken by the robot.

Consider first the result for 8 response directions ($n = 8$). The Bayesian and Dempster-Shafer maps are quite similar (not withstanding the fact that in the Bayesian map, 'unknown' cells appear as grey, whereas in the Dempster-Shafer map they appear white). These maps clearly show the various boxes, the hatstand and the chair in the test environment. A surprising feature of these maps is that multiple reflections (in which the ultrasonic pulse reflects off a number of surfaces

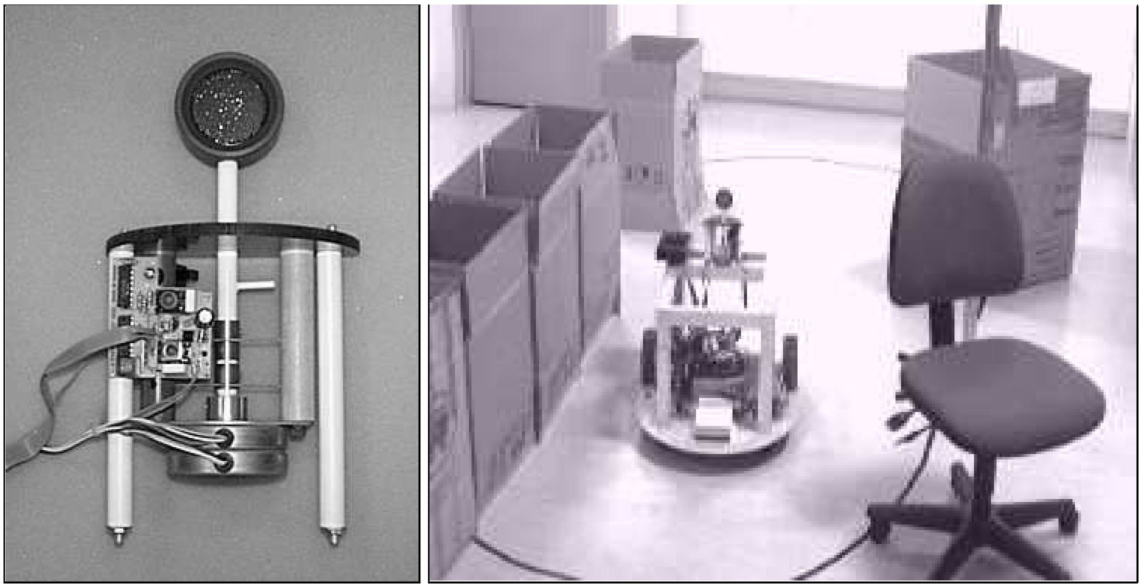


Figure 5.3: (a) J Edgar's rotating sonar head. (b) The test environment.

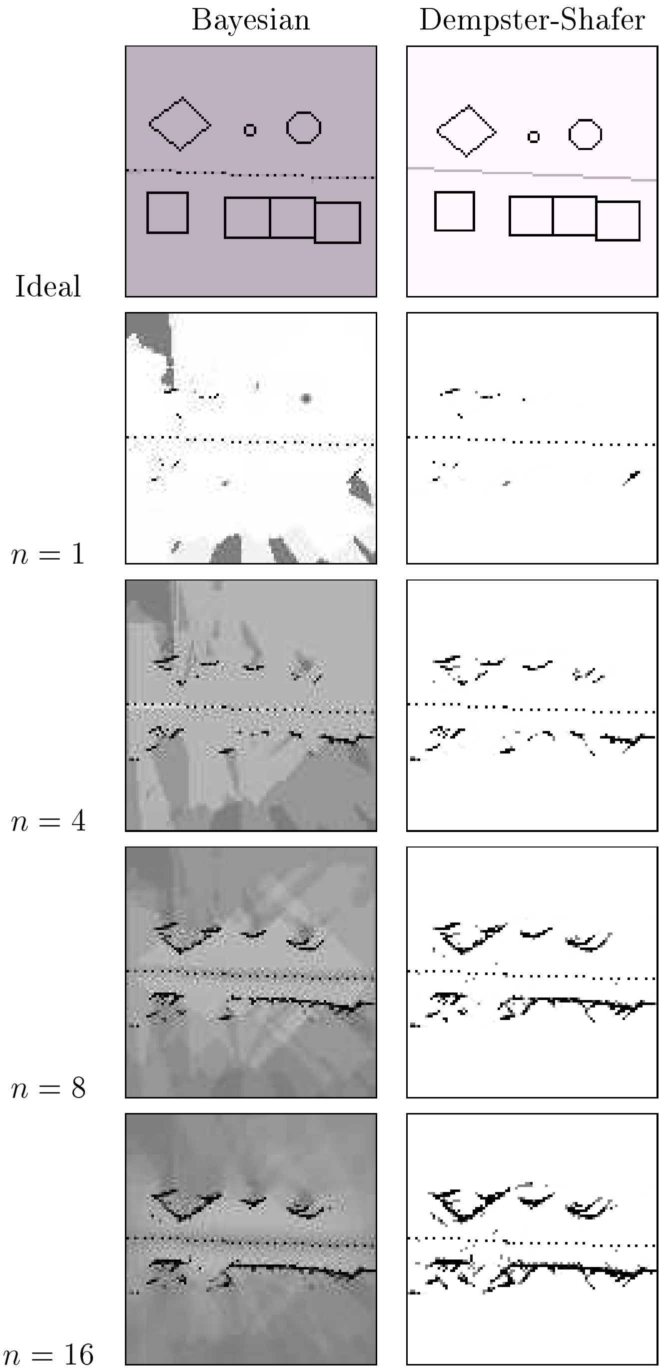


Figure 5.4: Occupancy maps obtained for the test environment. The dotted line shows the path taken by the robot.

before returning to the sensor) have *not* generated spurious features.

Comparing the $n = 8$ result with the $n = 1$ case, it can be seen that there is a sharp decrease in the quality of the maps as n becomes small. Note particularly the $n = 1$ case, where we are effectively ignoring the response behaviour and attempting to compute cell occupancy directly (the Bayesian $n = 1$ case corresponds more-or-less to the original occupancy grid method). The reason for this fall off is simple – the evidence obtained from multiple measurements is contradictory when n is small. Consider a smooth surface for example. A pulse whose angle of incidence on this surface is greater than about $40°$ will not be reflected back to the sensor. Consequently, this measurement will suggest that there is no response from the cell corresponding to the surface. Imagine that a second pulse is emitted, this time at an angle of incidence less than $40°$. This time a response is recorded. If n is large, these two measurements are consistent – the cell responds in one direction, but not in another. On the other hand, if n is small, a contradiction presents itself – two pulses were emitted in more-or-less the same direction (they correspond to the same direction 'bin'), but one generated a response and the other did not. In the extreme case, $n = 1$, the entire approach collapses.

On the other hand, close comparison of the $n = 16$ and $n = 8$ maps reveals that the $n = 8$ map is somewhat less 'noisy' (this applies to both the Bayesian and Dempster-Shafer methods). For large values of n, the chances that two measurements will fall into the same response direction 'bin' becomes very small. Consequently, for large n, there is very little combination of evidence occurring, and the quality of the map suffers. For this particular experimental configuration, the optimal value of n seems to be about 8.

Comparing the Bayesian result to the Dempster-Shafer result for $n = 8$, we can see that there is not a great deal of difference. As far as the implementation is concerned, however, the Bayesian method is slightly faster (it has fewer multiplications) and is more efficient in its use of memory (it requires $n+1$ values per cell, as opposed to $2n+1$ for the Dempster-Shafer method). Consequently, of the two methods, the Bayesian approach appears to be the best compromise between the quality of the map and the efficiency of the implementation. While it would be nice to conclude that the response grid approach yields better results that the standard occupancy grid approach (at least in specular environments), the results presented here do not warrant that conclusion. In particular, the results for $n = 1$, which should approximate the standard occupancy map approach, are significantly worse than those reported by other authors. Possibly, the differences are due to the different sensor models being used, and not the response grid approach per-se. Further experimentation needs to be done.

5.6 Conclusion

The experimental results in Section 5.5 clearly demonstrate the utility of the response grid approach in highly specular environments. There are, however, two important limitations in the work presented here.

Firstly, while the generated maps are very good at indicating which cells are *occupied*, they are very bad at indicating which cells are *unoccupied*. In the response grid approach, the only way to be sure that a cell is unoccupied is to view it from every possible direction. This is clearly impractical for robot navigation tasks. Unfortunately, for such tasks, it can be argued that knowing where obstacles *are* is less important than knowing where they *are not* (so long as the robot knows where obstacles are *not*, it can reliably avoid collisions).

Secondly, with only a single sonar sensor (which generates about 10 measurements per second), the rate at which the map is updated is extremely low. It takes five seconds for the head to make one complete sweep, and much can happen in these five seconds. High speed navigation, for example, is completely out of the question. To some extent, this problem can be addressed by replacing the sonar *head* with a sonar *ring* containing a number of sensors. This is the practice with many robot systems.

Of course, one way to avoid both of these limitations is to make use of an entirely different sensor modality. This was the solution adopted in Chapter 6, which described a method for building local maps using vision.

Chapter 6

Building Local Maps Using Vision

"There's no earthly way of knowing
Which direction they are going!
There's no knowing where they're rowing,
Or which way the river's flowing!
Not a speck of light is showing,
So the danger must be growing,
For the rowers keep on rowing,
And they're certainly not showing
Any signs that they are slowing..."

From *Charlie and the Chocolate Factory*, by Roald Dahl.

Abstract

This chapter looks at the problem of building environment maps using a single black-and-white camera. It shows how this camera can be turned into a 'virtual' range sensor, provided one makes certain assumptions about the environment the robot inhabits. The data produced by this 'virtual' sensor can be used to form an occupancy map *describing the robot's local environment.*

6.1 Introduction

One of the key difficulties facing any mobile robot is obtaining accurate, relevant and timely maps of the environment it inhabits. This chapter describes a vision-based mapping system that fulfils all these criteria, and is, in addition, fast, simple and relatively inexpensive (both computationally and financially). The system comes in two parts: a *visual range sensor* and a *visual mapping module*. The visual range sensor acts as a kind of 'virtual' range sensor: it takes as input a stream of images from the robot's camera and produces as output a stream of range-and-bearing measurements. The visual mapping module then fuses this data stream into a coherent grid-based representation of the environment, known as an *occupancy grid* [74, 76, 77, 172]. The motivation for this design arises from the observation that very good environment maps can be obtained by combining scanning laser range finders with grid-based data fusion techniques [128]. In this system, the laser range finder has been replaced with a vision-based 'virtual' range finder that produces comparable output, but at a fraction of the cost.

The Visual Range Sensor

The key part of this system is the visual range sensor. This 'virtual' sensor takes the stream of images produced by a single camera and translates it into a stream of range-and-bearing measurements. To do this, the sensor must solve two problems: it must determine the location of the obstacles *in an image*, and it must subsequently infer their location *in the world*. Clearly, in the general case, both of these problems are extremely difficult to solve. The visual range sensor therefore makes use of some assumptions about the environment the robot inhabits. Specifically, it is assumed that the robot inhabits an *indoor* environment such that:

- There are no overhanging obstacles on which the robot can 'bump its head'. This assumption allows us to reduce the general problem of finding an obstacles in an image to the much simpler problem of finding the *boundary* between an obstacle and the ground. So long as the robot stays outside this boundary, it will not collide with the obstacle. Note that the definition of what constitutes an 'overhanging' obstacle is dependant on the height of the robot. A table, for example, is an overhanging obstacle only if the robot is taller than the table (otherwise the robot will pass safely beneath it).

- The ground is untextured. This assumption makes it easier to find the obstacle/ground boundary, since it must correspond to the boundary of an untextured region.

- The ground is flat. If the robot is on a flat surface, there must exist a one-to-one correspondence between points in the image and points on the ground.

In computer vision, this is known as the *ground plane constraint.* Once the obstacle/ground boundary has been located *in the image*, the ground plane constraint can be used to determine the location of the corresponding boundary *in the world.*

Given that these assumptions hold, the visual range sensor can locate the obstacle/ ground boundary, first in the image and then in the world, using a very simple (and very fast) edge-detection algorithm. This algorithm is described in detail in Section 6.3.

The Visual Mapping Module

The role of the visual mapping module is to fuse the stream of range-and-bearing measurements generated by the visual range sensor to form an *occupancy grid* [74, 76, 77, 172]. As we saw in Chapter 5, the occupancy grid is a simple grid-based map in which each cell may be in one of two states: occupied (meaning there is an obstacle at this location) or unoccupied (meaning that there is no obstacle at this location). The occupancy state of each cell is determined using a simple Bayesian technique, in which multiple measurements from the visual range sensor are combined to determine the *probability* that the cell is occupied. The key benefit of using the occupancy grid approach in this context is that it increases the overall robustness of the system – the map building process will tend to 'wash-out' the errors produced by the visual range sensor (which is far from error-free).

Note that, as was the case with the sonar mapping module, the visual mapping module is responsible for building a *local* map, i.e. a map describing the area within a few meters of the robot. No attempt is made to generate a *global* map, which is a much more difficult task.

Chapter Structure

The structure of this chapter is as follows. Section 6.2 presents an overview of some related vision-based techniques, and provides further motivation for the particular technique described in this chapter. Sections 6.3 and 6.4 describe, in detail, the visual range sensor and the visual mapping module; while Section 6.5 presents a range of experimental results that show the visual mapping system in action. The chapter concludes with a discussion of the strengths and weaknesses of the visual mapping system in Section 6.6.

Note that an early version of the work described in this chapter can be found in [120].

6.2 Related Work

In this section, I will consider a number of related vision-based approaches. These approaches fall into four categories:

- Monocular – systems with one camera;
- Stereoscopic – systems with two or more cameras;
- Motion-based – systems that rely on motion of the camera;
- Structured-lighting – systems that rely on the projection of special lighting patterns.

This list by no means exhausts the set of possible vision-based approaches (see [126] for a more complete survey); it does however include the ones that are most important from the point of view of robot navigation.

Monocular Vision

Horswill [117, 118] describes an approach that is very similar to the one presented in this chapter. He assumes that the robot inhabits an environment with an untextured floor, and can therefore detect obstacles by looking for edges in the image. However, unlike the approach described in this chapter, no attempt is made to infer the location of the obstacles in *the world*. Instead, the robot avoids obstacles by monitoring their position *in the image*. Lorigo *et al* [159] extend Horswill's basic idea by adding *colour* information. Instead of just looking for edges, the robot employs three different obstacle detection *modules*. One of these modules looks for intensity gradients, the other two look for gradients in the RGB and HSV components of the image. The output of the three modules is combined by taking the median value. Experimental results indicate that this system is more robust using all three modules than it is using any one module (if one module fails, the other modules will usually compensate). As with Horswill, however, there is no attempt to infer the location of obstacles in the world; the robot avoids obstacles by monitoring their position in the image.

A similar idea is also presented by Cheng *et al* [51]; this time, however, a simple template matching scheme is used to distinguish between floor and non-floor regions. The robot is shown examples of the floor over which it is to travel, which are then stored as *templates*. At run time, the robot compares the stored templates against image and marks any mis-matches as obstacles. While this approach is quite attractive (it may even have the potential to handle textured floors, given a sufficiently large set of templates), the particular system demonstrated by the authors requires specialised hardware to achieve real-time performance. In addition, there is no attempt to determine the location of obstacles in the world; the robot avoids obstacles by monitoring their position in the image.

Crisman *et al* [61] describe a colour vision system (SCARF) for tracking roads and intersections for use in AGVs. There are two basic parts to this system: a pixel classifier, which classifies pixels into road and not-road based on their colour; and a geometric modeller, which attempts to fit a model of the road or intersection to the classified pixel data. The clever part of this system is the way in which it adapts automatically to changing road conditions: once the road has been located, the system extracts a set of sample road and not-road pixels from the original image; these sample pixels are used to classify pixels on the *next* frame. This approach could easily be applied to indoor robots, and represents a natural extension to the system described in this chapter.

A common feature of all of these monocular systems is that they work in *unstructured* environments. That is, they do not make strong assumptions about the geometric and structural properties of the environment (they make other assumptions instead). Also, with the exception of Crisman [61], none of these approaches build an intermediate representation of environment, such as occupancy map. Since they fall within the behaviour-based paradigm, vision is used to control the robot's behaviour directly.

Stereopsis

Many papers have been written on the subject of recovering range or structure from stereo image pairs (see [181] and [64] for a pair of reviews). Likewise, the use of stereo vision for robot navigation tasks has a very long history, dating all the way back to Stanford Cart [170]. Generally, these systems attempt to establish correspondences between the features found in a pair of images, given that the relative geometry of the two cameras is known. Commonly used features are points [171, 180] and line-segments [36, 148]. A number of authors have also described *self-calibrating* stereo vision systems [42, 103], in which the relative geometry of the cameras is only partially known (these systems usually rely on *motion* to provide extra information).

Trinocular stereo vision (i.e. stereo vision with three cameras) is also popular [14, 266]. The additional geometric constraints imposed by adding a third camera can simplify the correspondence problem significantly, leading to a substantial increase in system speed. Some commercial trinocular systems are now claiming near-real-time performance (10 frames/sec) using a Pentium MMX 166 MHz PC [199].

While stereo vision systems are well understood and readily implemented, they have a number of limitations when it comes to detecting obstacles in indoor environments. Firstly, indoor environments often contain large featureless surfaces, such as walls and doors. Using stereo, the robot can detect the *boundaries* of these obstacles, but cannot detect the obstacles *themselves*. Thus, for example, the robot cannot distinguish between an open door and a closed one (if the door is featureless). Secondly, stereo vision systems tend to be computationally expensive. One usually requires dedicated hardware or very fast computers to achieve near real-time performance. In contrast, the visual range sensor described in this chapter has been implemented on an 8-bit microcontroller, where it processes 10 frames/sec.

Motion

There is a substantial overlap between research on stereo (which involves one or more *fixed* cameras) and research on *motion* stereo (which involves one or more *moving* cameras). As is the case with stereo, there is much literature describing general techniques [45, 176, 248, 247, 254], but relatively few examples of motion-stereo applied to actual mobile robot systems. As with static stereo, these systems are usually feature-based, with popular features being points [267] and line-segments [154, 253].

The use of *optical flow* is also quite popular [55, 213]. Much of this research, however, is focused on controlling the motion of the robot directly, rather than on locating obstacles or building maps. The work of Srinivasan [230, 229] is particularly interesting in this respect, since it derives its inspiration from neurological studies of the visual-motor control mechanisms of bees.

The key limitation of motion-based approaches is, of course, that the robot has to move in order to detect obstacles. In addition, both the motion-stereo and optical flow based approaches suffer from the same problem: in indoor environments, features can be rather sparse. The optical flow based approaches, in particular, tend to require richly textured environments (having all visible surfaces covered with random dot paint would be ideal).

Structured lighting

There is a broad range of systems employing structured lighting to determine the range and structure of objects. Typically, these systems project some sort of pattern onto the object, and use one or more cameras to record this pattern. The structure of the object is the inferred from triangulation. Simple projection patterns include points [193] and lines [188, 228, 212]. These patterns must usually be scanned mechanically across the object of interest, which leads to relatively low frame rates for these systems. To increase the frame rate, more complex patterns are sometimes used. Unfortunately, this introduces a correspondence problem – one must match the observed features against the known features in the pattern. A common solution to this problem is to use *coded* structured light, in which the pattern carries additional information to simplify the correspondence problem. See [23] for a recent review of such techniques.

An interesting alternative to such coded light approaches is described by Ohya *et al* [187]. In this system, a regular array of dots is projected onto the floor in front of the robot. A camera mounted on the robot monitors these dots, and if a dot is displaced, the robot knows that there is an obstacle at this location. Thus, while the system is unable to extract full 3D information about the obstacle, it can at least detect the *presence* of an obstacle, and determine its location in 2D.

The simplicity of the structured lighting approach makes it very attractive for obstacle detection tasks. Indeed, the visual range sensor described in this chapter can be thought of a structured lighting technique, without the structured-lighting. There are two reasons I have choosen not to pursue a structured lighting approach. Firstly, the robot's camera must be fitted with a filter (usually an IR filter) so that only the structured light pattern remains visible. This prohibits the use of the camera for other purposes. Secondly, structured lighting introduces problems in multi-robot domains, where the patterns generated by different robots can cause interference. In contrast, the visual range sensor described in the chapter has been used very successfully in a number of multi-robot experiments [156].

6.3 The Visual Range Sensor

As noted in the Introduction, the task of the visual range sensor is to determine the range-and-bearing of any obstacles in the robot's environment. This 'virtual' sensor takes as input a stream of images from the robot's camera and produces as output a stream of measurements that indicate the range-and-bearing of the detected obstacles. To do this, the visual range sensor must solve two problems: it must determine the location of obstacles *in the image*, and it must subsequently infer their location *in the world*.

6.3.1 The Algorithm

In practice, the visual range sensor actually determines the location of the *boundary* between obstacles and the ground. Finding this boundary in an entire 512×512 or 256×256 image can be computationally expensive. Therefore, the visual range sensor adopts a 'sampling' technique, in which only some of the columns in an image are processed. The following algorithm is employed.

- Step 1. Apply a one-dimensional edge filter to each of the selected columns. Some of the edges located in this step will correspond to the obstacle/ground boundary; others will correspond to occlusion boundaries (i.e. to places where one object occludes another) and to features on the obstacles themselves. If the texture assumption described in the Introduction is satisfied, there will be no edges on the ground.

- Step 2. For each column in the image, select the edge that is closest to the *bottom* of the image. Most, but not all, of these edges will correspond to the obstacle/ground boundary.

- Step 3. Assume that all of the edges found in step 2 do, in fact, correspond to the obstacle/ground boundary. Using the ground plane constraint, compute the location of the corresponding obstacles in the world.

Clearly, this is an ad-hoc algorithm with little *theoretical* justification. However, so long as the assumptions listed in the Introduction hold, it is a very effective technique that can be implemented with minimal computational cost. In Section 6.3.2 we will look at some examples of the visual range sensor in action, and look at what happens in situations in which the assumptions *do not* hold.

The algorithm generates one *sample* – i.e. one range-and-bearing estimate – for each of the selected columns. An interesting feature of the algorithm is that the *rate* at which such samples are generated depends purely on the processing power available to perform the edge filter operations; it does not depend on the rate at which image *frames* are processed. Thus, for example, the visual range sensor could inspect 1 frame per second and generate 256 samples; or it could inspect 8 frames per second and generate 32 samples from each frame. In either case, the rate at which samples are generated is the same – 256 samples per second. This is an extremely useful feature: one is free to choose a frame rate that suits the environment the robot inhabits (dynamic environments generally demand higher frame rates than static environments), knowing that this choice will not affect the sampling rate.

Finding Obstacles in the Image

In Steps 1 and 2 of the algorithm, a one-dimensional filter is used to locate significant edges. While there are many, many filters that could be used for this purpose [47, 174], the particular filter used by the visual range sensor has been chosen purely for its speed.

In general, for some filter F, the filter response $R(n)$ of the n^{th} pixel on in a

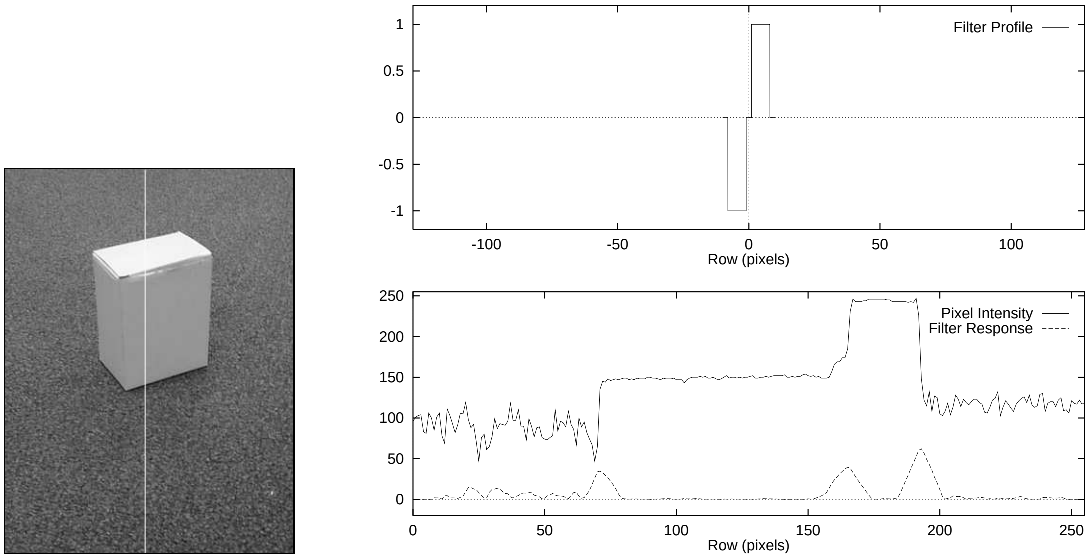


Figure 6.1: Filter profile and filter response for a sample image column; the sampled column is indicated by the line in the image.

column can be calculated as follows:

$$R(n) = \sum_{i=-\delta}^{+\delta} F(i) I(n+i), \tag{6.1}$$

where $I(n)$ is the intensity of the n^{th} pixel in the column and δ is the filter size. The particular filter used by the visual range sensor has the following form:

$$F(i) = \frac{1}{\delta} \begin{cases} -1 & -\delta \leq i < 0 \\ 0 & i = 0 \\ +1 & 0 < i \leq +\delta \end{cases} \tag{6.2}$$

For this filter, it is straight-forward to show that the response at $R(n+1)$ can be calculated from the response at $R(n)$ as follows:

$$R(n+1) = R(n) + \frac{1}{\delta} \left\{ -I(n) - I(n+1) + I(n-\delta) + I(n+\delta+1) \right\}. \tag{6.3}$$

Thus, instead of re-evaluating the filter response for each point (which involves inspecting 2δ pixels), the filter response can be computed incrementally (inspecting at most 4 pixels for each point). Use of this incremental technique yields a significantly faster algorithm when the filter size is large. To find *significant* edges, the filter response is thresholded. That is, an edge is regarded as significant only if the absolute value of $R(n)$ exceeds some preset *filter threshold*. It is assumed that the camera has automatic gain control, so that the average image intensity remains relatively constant.

The graph in Figure 6.1 shows the result of applying this filter to an image column. This particular column was taken from the middle of the image shown on the left. The top curve shows the raw pixel intensity values; the bottom curve shows the filter response. The first peak in the response curve indicates the location of the obstacle/ground boundary, while the second and third peaks correspond to the boundary between two facets and an occlusion boundary, respectively.

There are two free parameters associated with this part of the algorithm: the filter size and the filter threshold. Appropriate values for these parameters must be chosen empirically to suit the environment the robot inhabits. One can, however, write down a couple of general rules.

- If the filter size is too small, or the filter threshold is too low, the visual range sensor will detect many false obstacles (shadows or stains on the carpet will look like obstacles). These are known as *false positive errors*.
- If the filter size is too large, or the filter threshold is too high, the visual range sensor will fail to detect genuine obstacles (obstacles which are similar in colour to the ground will not be detected). These are known as *false negative errors*.

Thus, one must generally make a trade-off between detecting too many false obstacles (having a high false positive error rate) and failing to detect genuine ones (having a high false negative error rate). Note that the visual range sensor does not have to be perfect, any more than a sonar or laser range finder must be perfect. The visual mapping module, which fuses the data produced by the visual range sensor, will tend to 'wash-out' most of the errors produced at this stage.

Finding Obstacles in the World

Having determined the location of an obstacle in the image, the next step is to determine its location in the world; that is, to determine the range-and-bearing of the obstacle with respect to the camera. The transformation of a point from image coordinates to camera-centred coordinates can be done in a straight-forward manner by making using of the ground plane constraint – i.e. that fact the robot operates on an even surface. In this case, there must exist a one-to-one correspondence between points in the image and points on the ground. Hence, a point $(\hat{x}, \hat{y})$ in the image can be mapped onto a point (x, y) in the camera-centred coordinate system via a simple two dimensional transform:

$$\begin{aligned} x &= M_x(\hat{x}, \hat{y}) \\ y &= M_y(\hat{x}, \hat{y}) \end{aligned} \tag{6.4}$$

Furthermore, if we are using a pinhole camera (or if the camera is fitted with a lens such that it approximates a pinhole camera), it can be shown that this transformation must take the form of a *two dimensional projective transform* [99]. This transform is defined as follows:

$$\begin{aligned} x &= \frac{a_1\hat{x} + a_2\hat{y} + a_3}{d_1\hat{x} + d_2\hat{y} + 1} \\ y &= \frac{b_1\hat{x} + b_2\hat{y} + b_3}{d_1\hat{x} + d_2\hat{y} + 1} \end{aligned} \tag{6.5}$$

The transformation is controlled by eight coefficients $(a_1, a_2, a_3, b_1, b_2, b_3, d_1, d_2)$ whose values must be determined through a calibration procedure (this procedure is described in Section 6.3.3). Given Equation 6.5, the range and bearing (r, θ) of an obstacle can then be determined using elementary trigonometry:

$$\begin{aligned} r &= \sqrt{x^2 + y^2} \\ \theta &= \tan^{-1}(y/x) \end{aligned} \tag{6.6}$$

To save time, the visual range sensor does not actually use Equations 6.5 and 6.6 every time it wants to convert an image point to a range-and-bearing measurement. Instead, it pre-computes the values of (r, θ) for every point in the image and stores the values in a lookup table.

6.3.2 Some Examples

As a concrete illustration of the visual range sensor in action, consider the series of images shown in Figure 6.2. In this example, the robot Tigger has been placed on a carpeted surface with a simple box-shaped obstacle in front of it. The first three images show, in order: the view from the robot's camera; the output of the edge filter (step 1 in the algorithm); the output of the edge selection stage (step 2 in the algorithm). The final image shows the output of the visual range sensor, after the ground plane constraint has been applied (step 3 in the algorithm). The dots indicate the location of the detected obstacle. In this case, all of the assumptions have been satisfied and the algorithm correctly locates the obstacle.

As an example of what happens when the assumptions are not satisfied, or are only partially satisfied, consider the series of images shown in Figures 6.3 and 6.4.

In Figure 6.3, a chair has been placed in front of the robot. This object contains overhanging parts and consequently violates the first assumption. While the algorithm does find the obstacle in the image, it fails to correctly determine its location in the world. Specifically, it over-estimates the distance to the overhanging parts. Fortunately, in this case, the failure makes no *functional* difference, i.e. it does not affect robot's behaviour. The castors on the chair – which *are* correctly located – are too close together for the robot to pass between them; as a consequence, there is no chance that the robot will collide with this obstacle.

In Figure 6.4, the robot is sitting at the boundary between two dis-similar floor coverings – one surface is covered in carpet, the other in linoleum. This situation violates the texture assumption, and as a result, the robot will detect this boundary as an obstacle. This is perhaps the most severe limitation of this algorithm – the robot cannot move between parts of the environment with dis-similar floor coverings.

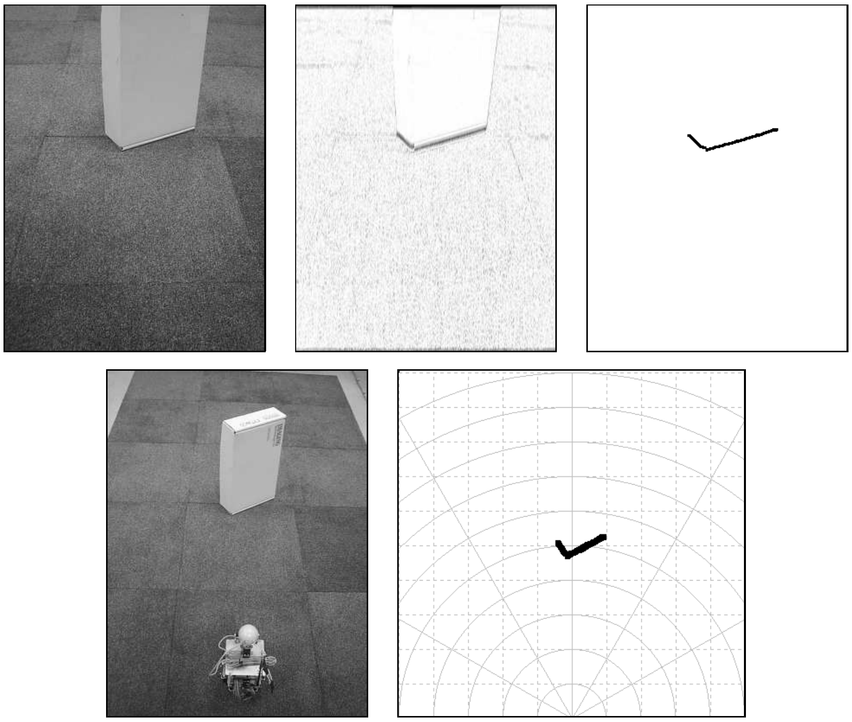


Figure 6.2: The visual range sensor in action, with a box placed in front of the robot. The top row shows: the view from the robot's camera; the result of applying a one-dimensional edge filter; the result after thresholding. The bottom row shows: an overhead view of the situation; the located obstacle, in camera-centred polar coordinates.

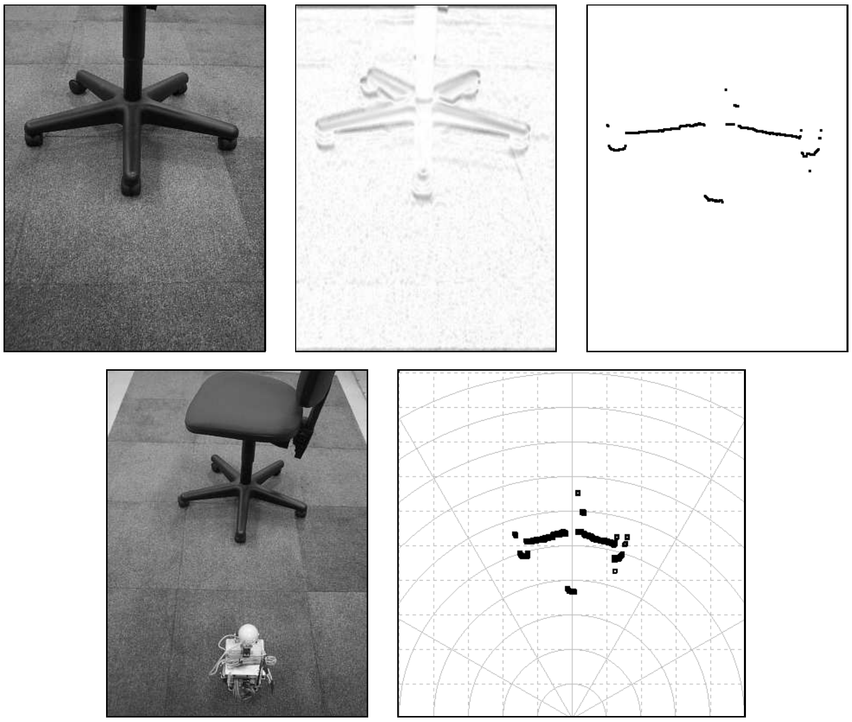


Figure 6.3: The visual range sensor in action, with a chair placed in front of the robot. The top row shows: the view from the robot's camera; the result of applying a one-dimensional edge filter; the result after thresholding. The bottom row shows: an overhead view of the situation; the located obstacle, in camera-centred polar coordinates.

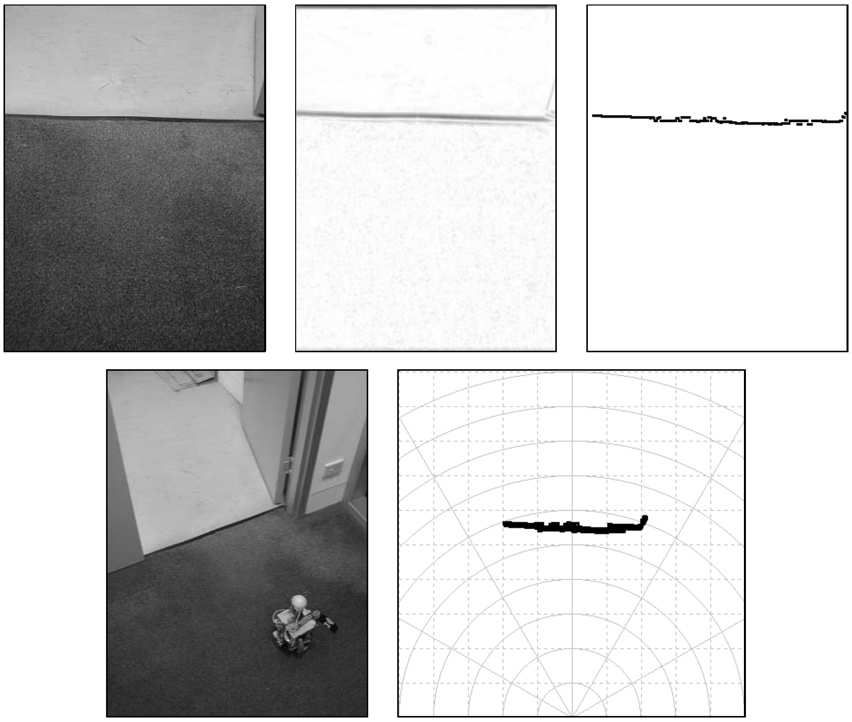


Figure 6.4: The visual range sensor in action, with the robot at a doorway. The top row shows: the view from the robot's camera; the result of applying a one-dimensional edge filter; the result after thresholding. The bottom row shows: an overhead view of the situation; the (mis-)located obstacle, in camera-centred polar coordinates.

Sometimes, however, this limitation can be used constructively – the robot can be 'penned' into a particular area, for example, by simply marking out the boundary of that area with an appropriately coloured tape.

Two other situations are worth noting:

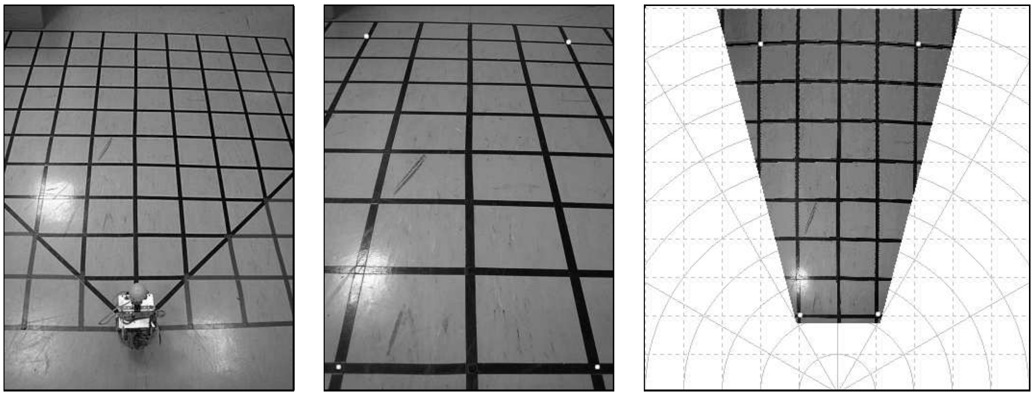
- The algorithm will not detect floor coloured obstacles. For example, it will not detect the boundary between a carpeted floor and a carpeted step. If such obstacles are common, an entirely different approach must be employed (one based on stereo or structured lighting, for example).

- The algorithm will not detect obstacles that occupy the entire field-of-view of the camera. For example, if the robot is very close to a wall, there will be no floor-obstacle boundary for the algorithm to detect. Fortunately, it is relatively easy to limit such occurrences through the use of an appropriate camera geometry (see Section 6.3.4).

6.3.3 Calibration

The two dimensional projective transform (Equation 6.5) can be calibrated by specifying four points whose position in both the image and camera-centred coordinate systems is known. Mathematically, the procedure is as follows. Let $\{(\hat{x}_1, \hat{y}_1), (\hat{x}_2, \hat{y}_2), (\hat{x}_3, \hat{y}_3), (\hat{x}_4, \hat{y}_4)\}$ denote a set of four calibration points defined with respect to the image coordinate system. Let $\{(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)\}$ denote the corresponding set of points defined with respect to the camera-centred coordinate system. Inserting these values into Equation 6.5, and after some re-arrangement, we get a set of 8 simultaneous linear equations.

$$x_1 = a_1\hat{x}_1 + a_2\hat{y}_1 + a_3 - d_1\hat{x}_1 x_1 - d_2\hat{y}_1 x_1$$



(a) (b) (c)

Figure 6.5: The camera calibration procedure. (a) Tigger on the calibration pattern. The lines are at 20cm intervals. (b) The view from the robot's camera. Four calibration points have been selected (the white dots). (c) The result of the camera calibration process. The entire image has been transformed into camera-centred coordinates.

$$\begin{aligned}
y_1 &= b_1\hat{x}_1 + b_2\hat{y}_1 + b_3 - d_1\hat{x}_1 y_1 - d_2\hat{y}_1 y_1 \\
x_2 &= a_1\hat{x}_2 + a_2\hat{y}_2 + a_3 - d_1\hat{x}_2 x_2 - d_2\hat{y}_2 x_2 \\
y_2 &= b_1\hat{x}_2 + b_2\hat{y}_2 + b_3 - d_1\hat{x}_2 y_2 - d_2\hat{y}_2 y_2 \\
x_3 &= a_1\hat{x}_3 + a_2\hat{y}_3 + a_3 - d_1\hat{x}_3 x_3 - d_2\hat{y}_3 x_3 \\
y_3 &= b_1\hat{x}_3 + b_2\hat{y}_3 + b_3 - d_1\hat{x}_3 y_3 - d_2\hat{y}_3 y_3 \\
x_4 &= a_1\hat{x}_4 + a_2\hat{y}_4 + a_3 - d_1\hat{x}_4 x_4 - d_2\hat{y}_4 x_4 \\
y_4 &= b_1\hat{x}_4 + b_2\hat{y}_4 + b_3 - d_1\hat{x}_4 y_4 - d_2\hat{y}_4 y_4
\end{aligned} \tag{6.7}$$

The eight free parameters $(a_1, a_2, a_3, b_1, b_2, b_3, d_1, d_2)$ can be found by solving this set of equations. Typically, this is done by re-expressing the equations in matrix form:

$$\begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \\ x_4 \\ y_4 \end{bmatrix} = \begin{bmatrix}
\hat{x}_1 & \hat{y}_1 & 1 & 0 & 0 & 0 & -\hat{x}_1 x_1 & -\hat{y}_1 x_1 \\
0 & 0 & 0 & \hat{x}_1 & \hat{y}_1 & 1 & -\hat{x}_1 y_1 & -\hat{y}_1 y_1 \\
\hat{x}_2 & \hat{y}_2 & 1 & 0 & 0 & 0 & -\hat{x}_2 x_2 & -\hat{y}_2 x_2 \\
0 & 0 & 0 & \hat{x}_2 & \hat{y}_2 & 1 & -\hat{x}_2 y_2 & -\hat{y}_2 y_2 \\
\hat{x}_3 & \hat{y}_3 & 1 & 0 & 0 & 0 & -\hat{x}_3 x_3 & -\hat{y}_3 x_3 \\
0 & 0 & 0 & \hat{x}_3 & \hat{y}_3 & 1 & -\hat{x}_3 y_3 & -\hat{y}_3 y_3 \\
\hat{x}_4 & \hat{y}_4 & 1 & 0 & 0 & 0 & -\hat{x}_4 x_4 & -\hat{y}_4 x_4 \\
0 & 0 & 0 & \hat{x}_4 & \hat{y}_4 & 1 & -\hat{x}_4 y_4 & -\hat{y}_4 y_4
\end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ b_1 \\ b_2 \\ b_3 \\ d_1 \\ d_2 \end{bmatrix} \tag{6.8}$$

and applying a standard matrix inversion technique [195]. Note, however, that some care must be taken in selecting the calibration points: if any three of the calibration points are collinear in either coordinate system, Equation 6.8 will have no solution.

In practice, the entire camera calibration process is largely automated. Figure 6.5 shows the three steps in the calibration process.

- The robot is placed on a special calibration pattern (with the camera carefully positioned above the origin) and is instructed to capture an image.

- A human operator selects four points and estimates their position in both the image coordinate system (by counting pixels) and the camera-centred coordinate system (by counting the divisions in the calibration pattern).

- Using these four points, a calibration program computes the 8 parameters that control the projective transform. The entire image is then mapped into camera-centred coordinates to test the accuracy of the calibration.

Figure 6.5(c) shows the result of mapping the original calibration image into camera-centred coordinates. Note the slight distortion in parts of this image – this distortion indicates that the real camera deviates somewhat from the pinhole model assumed by the projective transform. In this case, the distortion is minor – less than a few centimetres in the worst case. However, for cameras with short focal length lenses, the distortion can become much larger. For such cameras, the projective transform must be generalised to include higher order terms, which in turn necessitates the use of more calibration points.

6.3.4 Camera Geometry

The geometry of the camera determines three important characteristics of the visual range sensor – its minimum and maximum range, and its resolution.

Consider Figure 6.6, which shows the typical camera geometry. Clearly, the visual range sensor will only be able to detect obstacles that lie in the range $[r_{\min}, r_{\max}]$. Simple geometry tells us that these values are given by:

$$\begin{aligned} r_{\min} &= d \tan\left(\frac{\pi}{2} - \frac{\alpha}{2} - \beta\right) \\ r_{\max} &= d \tan\left(\frac{\pi}{2} + \frac{\alpha}{2} - \beta\right) \end{aligned} \tag{6.9}$$

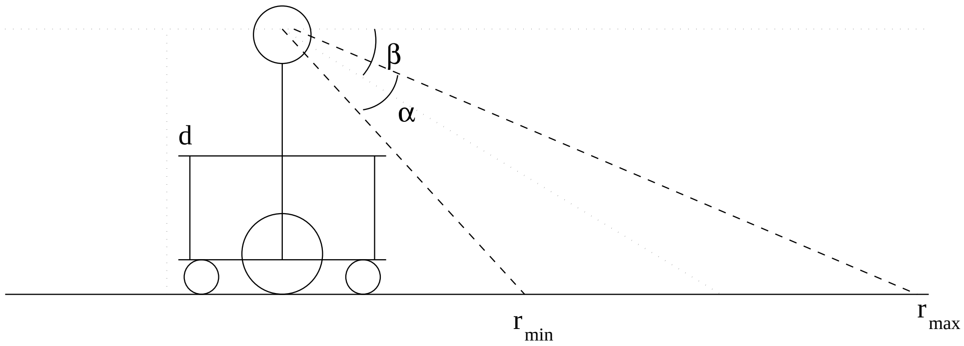


Figure 6.6: Camera geometry. The dashed-lines show the camera's field of view.

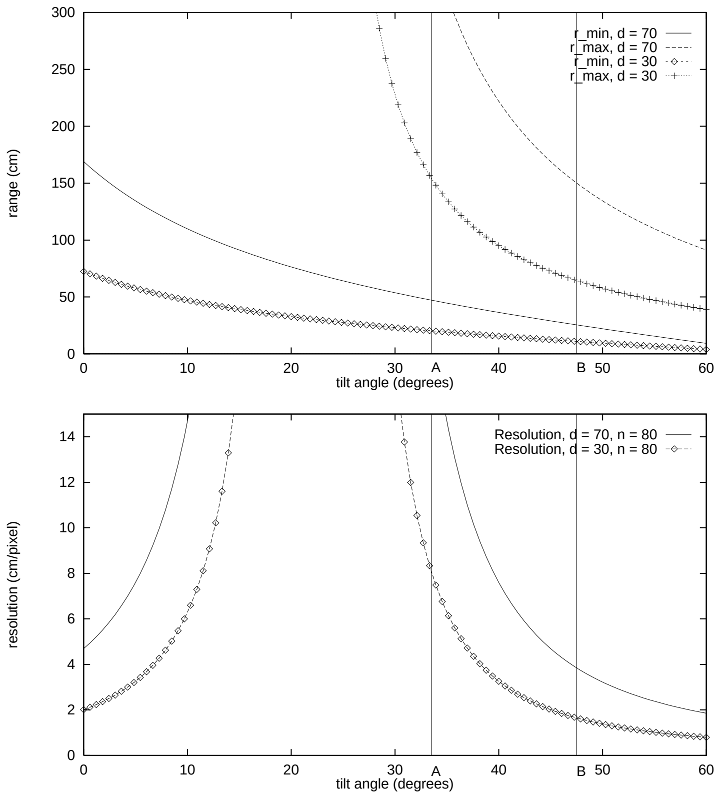


Figure 6.7: Range and resolution of the visual range sensor as a function of camera geometry ($\alpha = 45°, h = 80$). Note how the two configurations, marked A and B, have the same range, but different resolutions.

where α is a measure of the camera's field of view, d is the camera's height above the ground and β is the camera tilt angle. Note that α is generally a fixed property of the camera (or of the camera/lens system); for standard cameras, α is typically less than 45 degrees.

The camera geometry also dictates the resolution of the visual range sensor, i.e. the accuracy with which obstacles can be located. For the geometry shown in Figure 6.6, the resolution will be worst at the top of the image, where each pixel corresponds to a much larger region on the ground. Consequently, the *range* resolution of the visual range sensor is given by:

$$\Delta r = \frac{\alpha d}{h} \sec^2\left(\frac{\pi}{2} + \frac{\alpha}{2} - \beta\right) \tag{6.10}$$

where h is the height of the image, in pixels, and Δr measures the resolution in cm/pixel. One can also write down an approximate expression for the *angular* resolution of the sensor, which is simply:

$$\Delta\theta = \alpha / w \tag{6.11}$$

where w is the width of the image, in pixels, and $\Delta\theta$ is measured in degrees/pixel. For all realistic camera configurations, the angular resolution is better than $1°$/pixel.

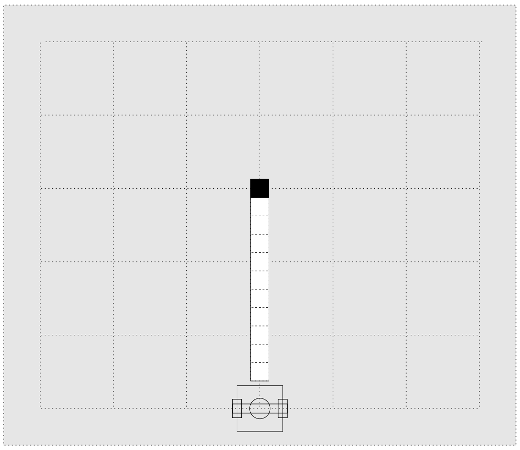
Ideally, we would like the visual range sensor to be able detect all obstacles with ranges from zero to infinity, and with unlimited resolution. In practice, of course, this is not possible. One must choose a camera geometry that represents a compromise between the competing demands of range and resolution. Figure 6.6 shows a set of graphs that make the situation clear. The first graph shows the minimum and maximum sensor range as a function of the camera tilt angle β. There are two sets of curves in this graph: one corresponds to a camera mounted at a height of 70cm, the other to a camera mounted at a height of 30cm. The second graph shows the resolution of the sensor as a function of camera tilt angle. Once again, there are two curves: one for a height of 70cm, another for a height of 30cm. Consider now the two configurations marked on these graphs. Both of these configurations have about the same minimum and maximum range, but they have very different resolutions. By mounting the camera at 70cm instead of 30cm, one obtains an almost two-fold increase in the resolution. Using graphs such as these, it is fairly easy to select a camera geometry that is appropriate to the situation. Alternatively, if the camera is mounted on a tilt head, one can select a number of different tilt angles at which to operate (unfortunately, one also has to calibrate the system separately for each particular tilt angle).

6.4 The Visual Mapping Module

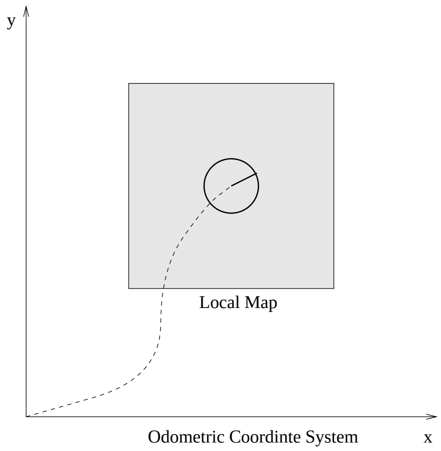
The visual mapping module fuses the stream of range-and-bearing measurements generated by the visual range sensor to form a *local occupancy map*. This is a simple grid-based map in which grid cells may be in one of three states: occupied (meaning there is an obstacle at this location), unoccupied (meaning that there is no obstacle at this location), and unknown. The occupancy state of each cell is determined using a simple Bayesian technique, in which multiple measurements are combined to determine the *probability* that the cell is occupied.

6.4.1 Mathematical Formalism

The occupancy grid approach can be expressed mathematically as follows. The occupancy of a cell at location (x, y) is represented by the state variable $\mathrm{Occ}(x, y)$,



(a)



(b)

Figure 6.8: (a) The sensor model. Only those cells lying between the robot and the detected obstacle are updated. (b) Relationship between the odometric coordinate system and the local map. The local map 'window' remains centred around the robot at all times.

which can have one of two values:

$$\mathrm{Occ}(x, y) = [\textit{occupied}, \textit{unoccupied}]. \tag{6.12}$$

The aim of the occupancy grid approach is to determine, for each cell, the *probability* that the cell is occupied. That is, to determine the probability of the proposition:

$$O_{x,y} : \mathrm{Occ}(x, y) = \textit{occupied} \tag{6.13}$$

Given some range-and-bearing measurement (r, θ), the probability that the cell at (x, y) is occupied can be determined using Bayes' Law:

$$p(O_{x,y} \mid r, \theta) = \frac{p(r, \theta \mid O_{x,y})}{p(r, \theta)} p(O_{x,y}) \tag{6.14}$$

where $p(O_{x,y})$ is the prior probability that the cell is occupied (usually set to 0.5 to indicate complete ignorance) and $p(r, \theta)$ is a normalisation constant.

The other term in Equation 6.14 is the *sensor model* $p(r, \theta \mid O_{x,y})$, which indicates the probability of obtaining the measurement (r, θ), given that the cell at (x, y) is occupied. The sensor model is the key element in the Bayesian formalism, since it encodes all of the robot's knowledge about the properties of the sensor. The sensor model used for the visual range sensor is as follows:

- If a cell has the same range and bearing as the measured value, this cell is probably occupied:

$$\begin{aligned} p(r, \theta \mid O_{x,y}) &= 0.9 \\ p(r, \theta \mid \neg O_{x,y}) &= 0.1. \end{aligned} \tag{6.15}$$

Note that $\neg O_{x,y}$ denotes the probability that the cell is *unoccupied*.

- If a cell has the same bearing as the measured value, but its range is less than the measured value, this cell is probably empty:

$$\begin{aligned} p(r,\theta \mid O_{x,y}) &= 0.1 \\ p(r,\theta \mid \neg O_{x,y}) &= 0.9. \end{aligned} \tag{6.16}$$

- For all other cells, the measurement tells us nothing:

$$\begin{aligned} p(r,\theta \mid O_{x,y}) &= 0.5 \\ p(r,\theta \mid \neg O_{x,y}) &= 0.5. \end{aligned} \tag{6.17}$$

Figure 6.8 depicts the sensor model in graphical form.

Note that this sensor model is much simpler than that used for ultrasonic range sensors (see Chapter 5). Compared to an ultrasonic range sensor, the visual range sensor has extremely good resolution in both range and bearing. So good, in fact, that the sensor model can entirely ignore the issue of limited resolution – the uncertainty associated with the sensor's finite resolution is almost always *less than* the cell size in the occupancy map.[1]

6.4.2 Implementation

In practice, the occupancy map is implemented as a simple two-dimensional array in which each cell represents a small region of the environment. For each cell, the map

[1] As is the case with sonar (see Chapter 5), it is possible to derive a much more sophisticated sensor model by considering both the properties of the sensor (in particular, the false positive and false negative error rates), and the environment in which it is to be used. However, the value of such a model is debatable, since the heuristic model described above works extremely well in a wide range of environments.

stores the probability $p(O_{x,y})$ that the corresponding region (x, y) of the environment is occupied. Cells are classified as being either *occupied*, *unoccupied* or *unknown* by applying a simple threshold operation to this probability value. Odometry is used to combine measurements taken from different robot locations to form a *local* map, i.e. a map describing the area within a few meters of the robot. This map can be thought of as a moving 'window' defined with respect to the odometric coordinate system. When the robot moves, the window moves with it. This situation is illustrated in Figure 6.8.

The map is updated in response to two different events.

- When the robot moves, the entire map is translated. As a result, some cells will 'fall-off' the edge of the map, while others will be 'added' to the opposite side of the map. The new cells are initialised to the default prior probability:

$$p(O_{x,y}) \leftarrow 0.5 \tag{6.18}$$

- When a range-and-bearing measurement (r, θ) arrives from the visual range sensor, the visual range sensor first determines which cells need to be updated. In this case, this is all of the cells that lie along the line joining the camera and the detected obstacle. These cells are updated using the incremental form of Bayes Law:

$$p(O_{x,y}) \leftarrow \frac{p(r, \theta \mid O_{x,y})}{p(r, \theta \mid O_{x,y})p(O_{x,y}) + p(r, \theta \mid \neg O_{x,y})p(\neg O_{x,y})} p(O_{x,y}) \tag{6.19}$$

  This equation assumes that successive measurements are statistically independent, i.e. that the errors associated with each measurement are due to random, not systematic, effects [28].

This update procedure is quite fast – on a Pentium 166 PC, a map containing 1600 cells can be updated in less than 100 μsec – which is fortunate, since the visual range sensor can easily generate measurements at a rate of around 600 samples per second.[2]

Note that, in any given image, there will generally be more than one image column that contributes to the occupancy state of a cell. Usually, this leads to a very reliable determination of the occupancy state of the cell. There is, however, at least one situation in which this is not the case. Consider a robot that is looking at the *corner* of an obstacle: some columns see the obstacle, while adjacent columns see past the obstacle. If these columns contribute to the same occupancy cell, the resultant state of the cell will be 'unknown'. Fortunately, this situation is very rare (since it requires a very particular alignment of obstacle, image and occupancy map). Furthermore, the effect on the map is negligible – in the worst case, corners of obstacles will be 'clipped' by one cell.

---

[2]If extra speed is required, one can make use of the more efficient *log-likelihood* formulation of Bayes Law [28]. In this formulation, the probability $p(O_{x,y})$ is replaced with a log-likelihood:

$$L(O_{x,y}) = \log \frac{p(O_{x,y})}{p(\neg O_{x,y})}. \tag{6.20}$$

Unlike probabilities, which takes values in the range $[0, 1]$, log-likelihoods take values in the range $[-\infty, +\infty]$. With this definition, Bayes Law becomes:

$$L(O_{x,y} \mid r, \theta) = L(r, \theta \mid O_{x,y}) + L(O_{x,y}) \tag{6.21}$$

where $L(r, \theta \mid O_{x,y})$ is the log-likelihood analogue of the sensor model. Since this form of Bayes Law requires only a single addition, as opposed to three multiplications and one division, it can be evaluated much more quickly.

6.5 Experiments

6.5.1 Properties of the Visual Range Sensor

The visual range sensor described in Section 6.3 has been implemented on two robots: J Edgar and Tigger. The two robots implement this sensor in very different ways, reflecting their very different capabilities. J Edgar is a fairly conventional robot, and is equipped in a fairly conventional manner: it has an ordinary monochrome camera connected to a frame grabber and a PC motherboard. The frame grabber acquires images measuring 256×256 pixels, which are then sampled using the algorithm described in Section 6.3. The robot Tigger, on the other hand, is somewhat unusual – it is equipped with a colour digital camera connected to an 8-bit microcontroller. This camera acquires images measuring 60×80 pixels, which are then processed *in their entirety* by the microcontroller. The output of both implementations is the same however – they both produce a stream of range-and-bearing measurements that are sent back to the host computer via a radio modem. The visual mapping module runs on this computer; the same module is used for both J Edgar and Tigger.

The properties of the visual range sensor, as implemented on J Edgar and Tigger, are shown in Table 6.1. This table describes five key characteristics of the sensor:

- Frame rate – the rate at which images are processed.
- Sampling rate – the rate at which range-and-bearing measurements are generated.
- Field of view – the region within which obstacles can be located.
- Resolution – the accuracy with which obstacles can be located.

	J Edgar	Tigger
Image size	256×256 pixels	60×80 pixels
Frame rate	10 frames/sec	10 frames/sec
Sampling density	32 samples/frame	60 samples/frame
Sampling rate	320 samples/sec	600 samples/sec
Field of view (range, angular)	[20cm, 140cm], $\pm 35°$	[20cm, 200cm], $\pm 16°$
Resolution (range, angular)	2cm, $< 1°$	10cm, $< 1°$
Filter size, threshold	10, 20	4, 20
Error rate (false positive, false negative)	8%, 9%	10%, 13%

Table 6.1: Characteristics of the visual range sensor for J Edgar and Tigger.

- Error rate – the rate at which both false positive and false negative errors are produced.

I will discuss each of these characteristics in turn.

Frame Rate

The frame rate is an important measure of how quickly the robot will respond to changes in the environment. On both J Edgar and Tigger, the frame rate is quite high – around 10 frames/second. This particular frame rate was chosen because it is just fast enough to capture the motion of a human walking at a normal pace; as a result, the robot can usually avoid collisions with pedestrians. Higher frame rates are possible (up to the limits imposed by the image acquisition hardware), but do not automatically lead to improvements in performance. This is partly due to the fact that the sampling density (i.e. the number of columns that are processed in each image) must drop as the frame rate increases (see Section 6.3). Consequently, the sensor is more likely to 'miss' an obstacle.

Sampling Rate

The sampling rate is an important measure of the amount of information being generated by the visual range sensor. Generally speaking, higher sampling rates lead to more reliable occupancy maps, since the visual mapping module has more information upon which to base its assessment of the occupancy state of a cell. Both J Edgar and Tigger have quite high sampling rates: 320 and 600 samples per second, respectively. This is more than enough to generate useful occupancy maps, even if the robot is travelling at quite high speeds (in excess of 30 cm/second). At first sight, the fact that Tigger has a higher sampling rate than J Edgar appears paradoxical.

Whereas Tigger is equipped with a simple 8-bit micro-controller, J Edgar has an entire 486 PC motherboard at its disposal. The disparity is partly explained by the difference in image sizes – Tigger is processing images that are one third the height of those processed by J Edgar – and partly by the difference in image acquisition hardware. The digital camera on Tigger is state-of-the-art and is very fast; the frame grabber on J Edgar is over 10 years old and is very slow.

Field-of-View

The field-of-view of the visual range sensor is determined principally by the camera geometry. Here, J Edgar has an advantage over Tigger – J Edgar is much taller, and consequently has a much greater field of view. The larger images used by J Edgar also give it much better resolution. In practice, resolution is not been a problem – being able to determine the location of obstacles to the nearest 10cm is more than adequate for most navigation tasks. In contrast, the limited field-of-view possessed by both robots *is* a problem – the robots may experience glancing collisions with objects that are just outside the camera's field of view. Consequently, it useful to pan the camera back-and-forth to increase the effective field of view; both robots are equipped with pan platforms for this reason.

Error rates

The visual range sensor will produce two kinds of errors: false positives, in which the robot detects an obstacle that is not really there, and false negatives, in which the robot fails to detect a genuine obstacle. These error rates are a key measure of the overall usefulness of the visual range sensor – if either rate is too high, the sensor will become ineffective.

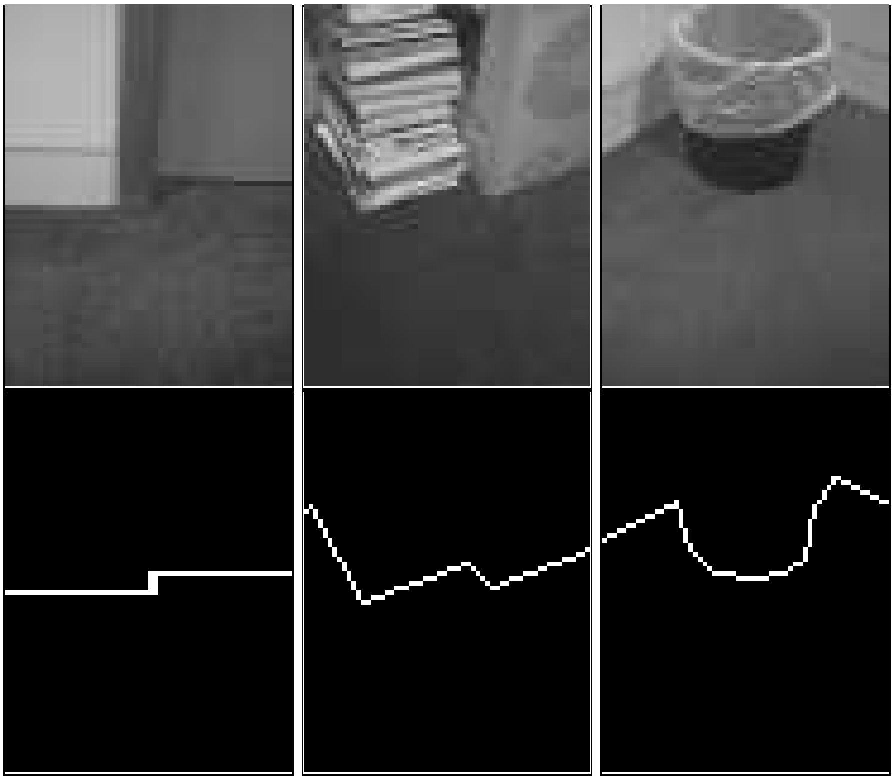


Figure 6.9: Some images from the test set. The top set of images show the view from the robot's camera, the bottom set shows the obstacle/ground boundary, as determined by a human operator.

The error rates quoted in Table 6.1 were determined using the following procedure.

- The robot is manually transported to a number of different locations, and instructed to take an image from each. This set of images forms a *test set*, and contains examples of all of the obstacles the robot is likely to encounter. Figure 6.9 shows three images taken from the Tigger's test set (the full set contains 12 images).
- From this set of images, the operator manually generates a set of 'ground truth' images, in which the location of the true obstacle/ground boundary is highlighted. Figure 6.9 shows a set of such images.
- Both sets of images are fed into an evaluation program, which computes the false positive and false negative error rates for the test set.

The evaluation program applies the first two steps of the algorithm described in Section 6.3 to every column of every image in the test set. The result is compared against the 'ground truth' image to determine whether or not the obstacle has been correctly located. The result is decreed to be a *false positive* error if the obstacle found by the algorithm is too close to the bottom of the image. The result is decreed to be a *false negative* error if the obstacle found by the algorithm is too close to the *top* of the image. Otherwise, the result is deemed to be acceptable. The overall error rates for the test set are computed by counting the number of columns which generate each type of error, and dividing by the total number of columns.[3]

[3] A similar procedure can be used to find an optimal value for the filter size and threshold parameters (see Section 6.3.1). Of course, one must then use a different set of images to test the system.

Inspecting Table 6.1, it can be seen that J Edgar and Tigger have similar error rates – around 10% for both false positive and false negative errors. While these rates are far from perfect, they are certainly low enough to produce useful occupancy maps, as I will show in the next section.

6.5.2 An Experiment with Tigger

Figure 6.10 shows a series of occupancy maps produced by Tigger during an experimental run. In this example, the test environment is an artificial one, made by laying out carpet tiles on a linoleum floor. The robot will detect the edge of the tiled region as an obstacle. The maps represent an area 400×400cm with a cell size of 5cm. At the start of each experiment, the robot builds up a map of its surroundings by turning in-place through 360 degrees.

Note the slight 'bending' effect that is apparent in some of these maps; this is a result of the cumulative drift associated with the odometric measurements used to determine the robot's position. This effect becomes more pronounced as the map becomes larger. Nevertheless, these maps are clearly adequate for the purposes of local navigation (i.e. obstacle avoidance).

6.5.3 An Experiment with J Edgar

Figure 6.11 shows series occupancy maps produced by J Edgar during a much longer experimental run. This time, the environment is a set of corridors in an ordinary office building (the environment has not been modified in any way). These maps measure 300×300cm with a cell size of 4cm. J Edgar is equipped with a panning head, which it uses to sweep the camera back-and-forth through an angle of about 60 degrees. This greatly increases the effective field-of-view. The construction of

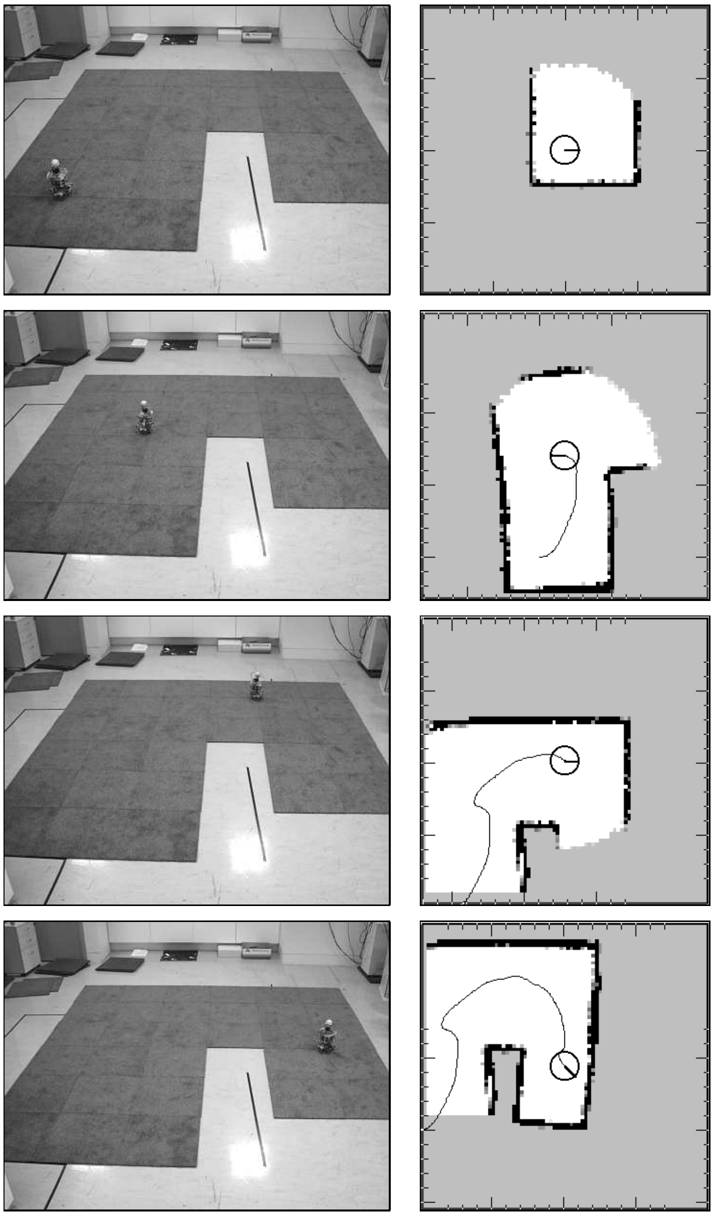


Figure 6.10: Occupancy maps produced by Tigger during an experimental run. Each of these maps is a 'snapshot' taken at a different point in time – the top map corresponds to the start of the run, the bottom map to the end. Dark pixels correspond to cells that are probably occupied, light pixels correspond to cells that are probably unoccupied; and grey pixels correspond to cells whose occupancy state is unknown.

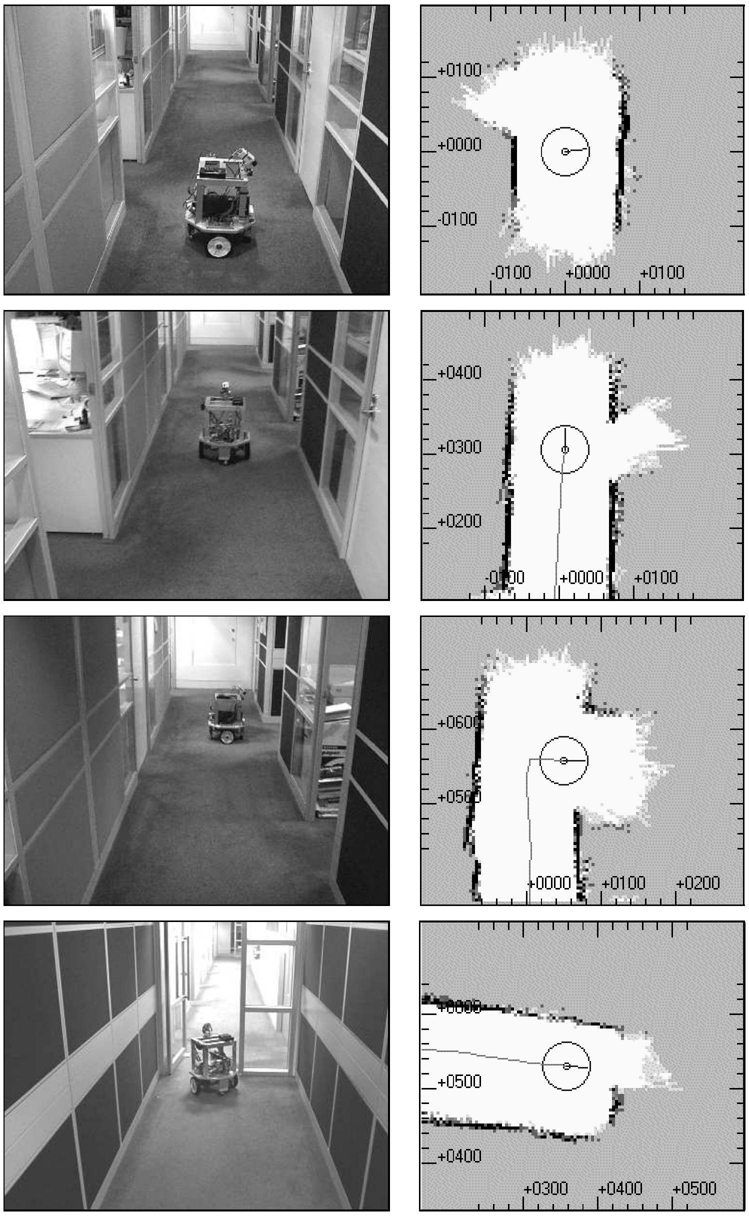


Figure 6.11: Occupancy maps produced by J Edger during a typical experimental run. Each of these maps is a ‘snapshot’ taken at a different point in time – the top map corresponds to the start of the run, the bottom map corresponds to the end. Dark pixels correspond to cells that are probably occupied, light pixels correspond to cells that are probably unoccupied; and grey pixels correspond to cells whose occupancy state is unknown.

the panning head (a spur gear system driven by a stepper motor) is such that the camera angle can be accurately determined.

As with the results for Tigger, the maps exhibit the 'bending' effect arising from odometric drift. The maps are, however, somewhat better than those produced by Tigger. This simply reflects the fact that J Edgar's odometry is more accurate than Tigger's. Note also that the maps are very 'clean', despite the fact that about 20% of the measurements generated by the visual range sensor are erroneous (either false positives or false negatives). Clearly, the Bayesian data fusion technique used by the visual mapping module is very effective at 'washing-out' these errors. The final significant feature of these maps is the way in which walls, doorways and intersections can be clearly resolved. In Chapter 8, I will describe a mechanism for extracting these landmarks from the local occupancy map.

6.6 Conclusion

The visual range sensor described in the chapter is extremely simple. Nevertheless, when combined with the visual mapping module, it produces very good results. It is certainly fast enough to support navigation at a relatively high speed (Tigger is routinely operated at a speed of 30cm/sec, with a peak speed of 50cm/sec). It does, however, have one very pronounced limitation: it can only be used in environments with untextured floors. In the future, it should be possible to overcome this limitation by using *colour* information to improve the segmentation of the image into floor and non-floor pixels.

The approach described in this chapter has shown itself to be quite robust to the presence of moving obstacles (in the form of both humans and other robots). This is despite the fact that it was never designed to handle dynamic environments. So long as moving obstacles remain within the robot's field-of-view, they appear within the occupancy grid as moving clusters of occupied cells. However, if the robot shifts so that the obstacle is *outside* the robot's field-of-view, some very strange artefacts can arise (since the obstacle is assumed to have stopped at its last recorded position). If one were serious about handling dynamic environments, the standard occupancy grid approach would need to be extended, perhaps with the addition of a 'forgetting' factor. Such extensions are, unfortunately, beyond the scope of this thesis.

Chapter 7

Local Navigation

> "I try to plan, in your sense of the word, but that isn't my basic mode, really. I improvise. It's my greatest talent. I prefer situations to plans you see..."

from *Neuromancer*, by William Gibson.

Abstract

This chapter describes the local navigation module, which is responsible for local navigation tasks. For this, the module makes use of a novel approach, which explicitly takes into account the robot's dynamics*, as well as its kinematics. As a result, the motion produced by the local navigation module is both fast and smooth.*

7.1 Introduction

Within the navigation system described in this thesis, the overall navigation task is divided between the local and global layers. While the global layer is responsible for long term navigation problems, which may involve moving the robot many tens of meters, the local layer is responsible for short-term navigation problems, which will involve moving the robot a few meters at most. As noted in Chapter 3, this division of responsibilities has a double benefit: the global layer can treat navigation as a fairly abstract planning problem, in which all the messy details of actually detecting and avoiding obstacles is delegated to the local layer, while the local layer can treat navigation as a reactive problem, in which no long-term planning is required. Within the local layer, it is the local navigation module that is responsible for local navigation tasks.

The Local Navigation Module

The local navigation module has two basic roles: to avoid obstacles, whilst simultaneously seeking some nearby goal. The obstacles it avoids are those present in the local occupancy map, and the goals it seeks are those given to it by the global layer. The local navigation module has been designed to support relatively high speeds – up to 50cm/sec. At these speeds, many of the standard navigation techniques (which are reviewed in Section 7.2) fail to produce reliable behaviour. The problem with these techniques is simple: they fail to adequately account for the robot's *dynamics*. Specifically, they fail to allow for the fact that the robot is capable of only *limited* acceleration. The local navigation module therefore makes use of a novel approach, one in which the dynamics of the robot are considered in an explicit manner. For reasons that will become apparent, this approach is called *dynamic utility fusion* (DUF for short).

The local navigation module has three components: a set of *constraints*, a set of *behaviours* and an *arbiter*.

- The set of constraints determines which *sequences of actions* are physically possible. The robot's maximum speed and acceleration, for example, are expressed as constraints.

- The set of behaviours expresses the *utility* of different sequences of actions. The avoid-obstacle behaviour, for example, indicates that it is generally undesirable to collide with an obstacle. The move-to-goal behaviour, on the other hand, indicates that moving towards the goal is generally preferred over moving away from it.[1]

[1] The term 'behaviour' is something of a misnomer in this context. The 'behaviours' in the

- The arbiter looks for the most desirable sequence of actions that satisfies all of the constraints.

A key aspect of this approach is that it considers the effect of *sequences* of actions, rather than the effect of *individual* actions. That is, the local navigation module understands that, at high speeds, the action the robot performs *now* must be conditioned by the actions it *intends* to perform later. Consider, for example, a robot which is approaching a wall at the relatively low speed of 20cm/sec. If the robot has a maximum rate of deceleration of 50cm/sec/sec, it will take only 0.4 seconds to come to a complete stop; in this time, it will travel 4cm. Consequently, the robot does not need to take any action until it is very close to the wall. Now imagine that the same robot is travelling towards the wall at 100cm/sec. The robot will take at least 2 seconds to come to a complete stop, and in this time, it will travel 100cm. The robot must start decelerating a full 2 seconds before it reaches the wall. At 20cm/sec, the robot does not need to plan more than a fraction of a second ahead; at 100cm/sec, the robot must plan at least 2 seconds ahead. For a detailed analysis of the requirements for high speed robot navigation (including the issue of finite look-ahead), see Kelly et al [132, 133]

The second key aspect of the DUF approach is the manner in which the arbiter selects sequences of actions for the robot to perform. The arbiter must search the space of all possible sequences of actions (subject to the set of constraints) to find the one with the highest overall utility (according to the set of behaviours).

---

local navigation module do not propose specific sequences of actions; they merely comment on the various alternatives. Hence, they would be better described as ‘desires’. Nevertheless, the term ‘behaviour’ will be retained, since it is consistent with the current usage of this term in the literature [9, 39, 204, 203].

Unfortunately, the space of all such sequences is very large, and the arbiter has very little time in which to make a decision (a few hundred milliseconds, at most). Since the arbiter cannot possibly search the entire space in this time, it makes use of a *stochastic search strategy*. This strategy involves evaluating a small subset of randomly chosen sequences, and executing the best one found in the time available. As I will show in Section 7.4, this strategy will almost never find the *best* sequence, but it will almost always find a sequence that is *good enough*.

There are a number of advantages to the DUF approach.

- Dynamic constraints can be expressed easily and explicitly. One can, for example, limit both the robot's speed and acceleration.
- Dynamic behaviours can also be expressed. One can, for example, construct a behaviour that expresses the utility of moving forwards over moving backwards (while still allowing the robot to back away from nearby obstacles).
- There is a high degree of modularity – it is easy to add both new constraints and new behaviours.

Finally, and most importantly, the resultant motion is both fast and smooth: the robot will decelerate into corners and accelerate out of them again, it will swerve to avoid collisions, and it will decelerate smoothly to a stop at the goal.

Chapter Structure

The remainder of this chapter is structured as follows: Section 7.2 presents a review of related navigation techniques; Section 7.3 describes the local navigation module in detail, and Section 7.4 presents experimental results obtained with the robots

J Edgar and Tigger. Finally, Section 7.5 presents some conclusions and indicates directions for further research.

7.2 Related Work

Local navigation is a very well-studied area. There are, it seems, almost as many local navigation/obstacle avoidance techniques as there are mobile robots. This review does not attempt to present an exhaustive survey of these techniques; instead, it has much more modest objectives: to identify the major paradigms in this area, and to indicate the sources of inspiration for the dynamic utility fusion approach.

The navigation techniques described in this review are divided into four categories:

- Potential field methods
- Command arbitration methods
- Fuzzy logic and neural network methods
- Analytic methods

While these categories by no means cover the full range of possible techniques, they do cover those that are most closely related to the dynamic utility fusion approach.

Potential Field Methods

Among the most popular techniques in use today are those based on artificial potential fields [8, 9, 15, 113, 134]. This technique, first popularised by Khatib [134] and Krogh [149], is based on an analogy with electrostatics: the robot is imagined to be a charged particle which is repelled from obstacles and attracted to the goal. The potential field approach can be framed in one of two ways: one can either construct a scalar potential field, and determine the imaginary force on the robot by computing the gradient of this field; or one can determine the imaginary force directly, by summing all of the forces acting on the robot. While the later case tends to be somewhat more efficient from a computational viewpoint, the former case admits the use of more complex potential fields. One can, for example, make use of harmonic functions [264] or distance transforms [127], both of which have the useful property of having no local minima. Unfortunately, these more complex fields are also expensive to compute; their use for real-time navigation has therefore been somewhat limited.

Part of the reason for the popularity of the potential approach is its simplicity and its versatility. It can, for example, be used for both sensor-based and model-based navigation: in the former case, the potential is generated by the obstacles that the robot has *sensed*, in the latter, the potential is generated by the obstacles *in the model*. Hybrid approaches, in which both sensor and model data are used, are also possible.

Unfortunately, potential fields suffer from two key problems (see [144] for an analytic treatment):

- The robot may become trapped in local minima (if one is using simple potentials).
- The robot may experience oscillations in cluttered environments or narrow corridors.

The first of these problems arises mainly from the fact that, for the sake of speed, most robots rely on very simple potentials, which exhibit local minima. These robots often become trapped behind obstacles or in dead-ends. Various solutions have been proposed for this problem. Arkin [8, 9] adds a random 'noise' component to the field, the purpose of which is to drive the robot out from behind obstacles. A more comprehensive solution is described by Balch and Arkin [15], in which the robot is repelled from places that it has already visited. This not only drives the robot out from behind obstacles, it also drives the robot out of dead-ends.

The problem of oscillations is a more serious one. Oscillations arise as a direct consequence of the failure of the potential field approach to account for the *dynamics* of the robot. Generally, the robot is treated as a massless particle, with zero momentum and infinite acceleration. Since this is most definitely not the case, the robot can find itself 'bouncing' back and forth between obstacles, as its own momentum causes it to overshoot the ideal path. At low speeds, such effects are rarely observed; as the robot's speed increases, however, they become increasingly common. There have been many modifications of the potential field approach that have been aimed, directly or indirectly, at addressing the problem of oscillations. Some of these are fairly ad-hoc: both Borenstein *et al* [30] and Holenstein *et al* [113] describe modified versions of the potential field approach in which a velocity dependant damping term is added. The effect of this term is to increase the repulsion from obstacles that lie in the direction of the robot's motion, and decrease the repulsion from other obstacles. This allows, for example, the robot to travel down a narrow corridor at high speed, yet still brake to a halt when it reaches a dead-end. Guldner *et at* [104, 105] take a different approach, and attempt to address the problem from the point of view of control theory. In [105], they look at how a robot with dynamic constraints may employ a *sliding mode controller* to 'track' a potential. The advantage of this technique is that it offers a rational procedure for transforming the potential field into actuator commands (as opposed to the heuristic rules used in [30] and [113]). Unfortunately, none of these approaches are able to offer a guarantee that the robot will not collide with an obstacle as a result of dynamic effects.

The next two approaches do not, strictly speaking, fall into the category of potential field approaches. Nevertheless, the general concepts they employ are sufficiently similar to potential field approaches that they are included in this part of the review. Furthermore, these approaches are specifically designed to address the problem of robot dynamics.

Borenstein *et al* [31] describe a *vector field histogram* approach (VFH for short). The starting point for this approach is a simple occupancy grid, from which one calculates a polar histogram around the robot's location. Nearby obstacles produce peaks in the histogram; open spaces produce valleys. Hence, by steering towards valleys in the histogram, the robot can avoid collisions. In this form, the VFH approach does not consider the robot's dynamics; however, Ulrich and Borenstein [251] describe an extended approach (VFH+) that overcomes this limitation. The original polar histogram is subject to further processing steps, including a 'masking' stage. The 'masked' polar histogram shows which directions of motion are possible *at the current speed*. If no directions are possible, the histogram must be re-generated for a lower speed. Ultimately, if the robot is unable to find any open directions at any speed, it comes to a complete stop.

Fox *et al* [93, 244] describe a *dynamic window* approach (DWA for short). Instead of attempting to control the robot in position space, the approach attempts to control the robot in *velocity space*, i.e. the space made up of the robot's linear and angular velocity. Since the robot is capable of only finite accelerations (either because of torque limitations in the motors, or because the robot will topple over at high accelerations), the authors define a 'dynamic window' in the velocity space that includes all of the velocities that can be reached by the robot in the next time interval. An objective function is applied to every velocity within this window to determine the optimal velocity. The objective function considers such issues as the distance to the nearest obstacle and the progress being made towards the goal. The authors report very good results, with speeds of up to 95cm/sec in cluttered environments.

An interesting extension to this approach (μDWA) is given in [94]. In its original form, the dynamic window approach relies on sensor data to determine the location of obstacles. In this latter form, obstacles that the robot *expects* to be present, based on a model of the environment, are included. In this way, the robot can avoid obstacles that have not been detected. It does, of course, assume that the robot has a global model of the environment and is able to determine its location with respect to this model. Nevertheless, very good results are reported (the system was trialed in a heavily populated museum).

The dynamic utility fusion approach described in this chapter is strongly influenced by the dynamic window approach. Indeed, to some extent it can be viewed as generalisation of this approach. However, unlike the dynamic window approach, which considers only the next time interval when choosing actions, the dynamic utility fusion approach considers extended *sequences* of actions that may take several seconds to execute. In this respect, it is somewhat similar to the approach described by Singh and Kelly [227], which constructs plans in the robot's action space.

Unfortunately, one corollary of this is that the dynamic utility fusion approach is computationally expensive; more expensive, in fact, than any of the potential field techniques described in this section. Nevertheless, as I will show in Section 7.4, the dynamic utility fusion is fast enough to support real time navigation. Furthermore, this approach has the advantage of being much less prone to problems associated with local minima and oscillations, and can guarantee that the robot will not collide with any obstacle that has been detected by the robot's sensors.

Command Arbitration

The basic idea behind *command arbitration* is to facilitate modularity in the design of robotic systems. Rather than having one huge sense-plan-act module, which is responsible for all aspects of the robot's behaviour, the robot has a set of *modules*, each of which is responsible for certain tasks or behaviours. Each of these modules takes some form sensory data as input, and produces as output some form of command for the robot to perform. Command arbitration is used to fuse these outputs into a single command for the robot to perform. While there are many ways in which this command arbitration can be carried out (see [8, 9, 37, 98, 169, 205, 204, 203] for example), I will consider only a few key examples.

In Brooks' original Subsumption Architecture [37], modules are formed into *layers*, with each layer representing a new level of *competence* for the system. Lower layers express simple behaviours, such as 'avoid-obstacle', while higher layers express more complex behaviours, such as 'follow-wall'. Command arbitration is hierarchical: higher layers are able to override the outputs of lower layers, but not vice-versa. Consequently, the Subsumption Architecture could more accurately described as a *command selection* scheme. It is, as a result, comparatively inflexible.

In Arkin's Motor Schema approach [9, 8], control is divided between a number of different *motor schemas*. Each schema expresses a different behaviour, such as 'avoid-obstacle' or 'stay-on-path'. Each of these schemas produces an output vector, indicating a desired direction of motion. Command arbitration between various motor schemas is achieved by performing a simple summation of the vectors produced by each schema. This is, unfortunately, not a particularly good mechanism – vectors may add constructively in such a way that the robot is drawn too close to an obstacle [15], or they may add destructively in such a way that the robot becomes indecisive (like Buridan's Ass, caught between two equidistant piles of hay).

A much better solution is described by Rosenblatt [205] with his Distributed Architecture for Mobile Navigation (DAMN). Here, the system is made up of a set of behaviours and a central command arbiter. Each behaviour may vote either for or against an action; the arbiter collates the votes and executes the winning action. A key difference between the DAMN approach and both the Subsumption Architecture and Motor-Schema approaches is that the 'behaviours' in DAMN do not indicate any specific action for the robot to perform. Instead, they are able to vote for a range of actions. This makes the approach both flexible and powerful, as demonstrated by the fact that it has been used extensively to control both on- and off-road vehicles. There is even a fuzzy logic version of DAMN [263], for which the authors claim improved results.

The dynamic utility fusion approach described in this chapter is heavily influenced by Rosenblatt's DAMN architecture. From it, the dynamic utility fusion approach takes the idea of having multiple behaviours connected to a central arbiter. In addition, while it lacks the relatively rich voting mechanism of DAMN (a much simpler summation-of-utilities mechanism is used), the dynamic utility fusion approach considers much longer sequences of actions, and adds explicit consideration of the robot's dynamics.

Fuzzy and Neural Approaches

Local navigation systems based on fuzzy logic and neural networks have been widely explored [27, 49, 50, 250] (see Saffiotti [211] for a more complete survey). One advantage of these approaches is that the system can be *trained* to carry out certain tasks, such as obstacle avoidance. For example, Beom and Cho [27] describe a navigation system consisting of two fuzzy modules: an obstacle avoidance module and a goal-seeking module. Each of these modules contains a set of fuzzy rules that map the input vector (a set of sonar readings) onto an output vector (a motor command). The two modules are trained using a reinforcement learning scheme, so that the system can learn directly from experience. Interestingly, a non-fuzzy command selection routine is used to determine which of the two modules is in control of the robot at any given time.

While fuzzy logic and neural network approaches tend not to consider the robot's kinematic or dynamic constraints in an explicit fashion, their ability to *learn* implies that such constraints may be encoded *implicitly* into the fuzzy rule set or neural network. One drawback, however, is that it can take a great deal of time to train these systems for the full range of conditions the robot is likely to encounter.

Analytic Approaches

A number of authors have tackled the local navigation problem from the perspective of classical mechanics and control theory [5, 13, 81, 82, 95, 153, 223, 222]. These authors have considered the problem of generating an optimal path for the robot, whilst taking into account the robot's kinematic and dynamic constraints. In this case, 'optimal' can have a number of meanings – it may involve minimising the amount of time required to reach the goal, or it may involve maximising the 'smoothness' of the generated path [81, 153]. Unfortunately, most of these approaches assume that the robot has a complete world model, which makes them inappropriate for sensor-based navigation.

An exception to this rule is the work of Shkel and Lumelsky [5, 223, 222], who consider the problem of generating optimal trajectories using only sensory data. In this approach, the robot generates *canonical* trajectories, which represent time-optimal paths in the absence of obstacles. The trajectories are then modified to create *near*-canonical trajectories, which take into account any obstacles that are visible to the robot's sensors. The trajectories take into account both the robot's kinematics and its dynamics. Despite the fact that these trajectories are found by solving a quite complex optimisation problem, the authors claim that the system can run in real time (less than 20ms). Since most of the results presented to date are for simulated data only, it is not clear how this approach compares to the dynamic utility fusion approach, either in terms of its computational complexity or the results it produces.

7.3 The Local Navigation Module

As was noted in the Introduction, the local navigation module has three key components: a set of constraints, a set of behaviours and a command arbiter.

- The set of constraints determines which sequences of actions are allowable.
- The set of behaviours determines the utility of various sequences of actions.
- The arbiter searches through the space of all possible sequences to find the most desirable sequence that satisfies all of the constraints.

In the following sections, each of these components is discussed in detail. First, however, we must consider the relationship between actions, sequences of actions and trajectories.

7.3.1 Actions, Sequences of Actions and Trajectories

Most of the behaviours employed by the local navigation module do not evaluate the utility of sequences of actions directly. Instead, they evaluate the utility of the *trajectory* that results from a particular sequence. A trajectory describes the evolution of the robot's position, velocity and acceleration as a function of time. A key assumption, therefore, is that the relationship between sequences of actions and trajectories is known – that is, that the robot's motion is *predictable*. Mathematically, we assume that we have some model of the robot such that

$$\eta = \mathrm{Model}(\zeta) \tag{7.1}$$

where ζ is a sequence of actions and η is the resultant trajectory.

The exact nature of what constitutes a sequence of actions ζ is determined by the physical construction of the robot. J Edgar and Tigger have the same basic construction: a pair of independent drive wheels and a pair of passive idler wheels. This arrangement is depicted in Figure 7.1. The motion of the robot is determined by the speeds (s_1, s_2) of the two drive wheels. If these speeds are identical, the robot will drive in a straight line; if these speeds are identical in magnitude, but opposite in sign, the robot will turn in place. Other values will cause the robot to execute some curved path. Both robots have PID controllers that will – to a very good approximation – maintain a given wheel speed. For these robots, a sequence of actions ζ therefore takes the form of a function which describes the time evolution

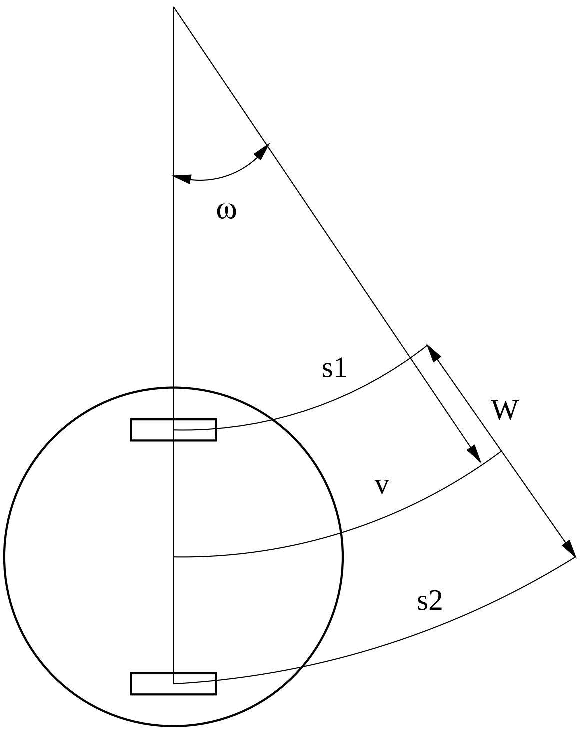


Figure 7.1: Robot kinematics.

of the wheel speeds (s_1, s_2). We write this as:

$$\zeta(t) = [s_1, s_2] \tag{7.2}$$

where $\zeta(t)$ is defined on some time interval $[t_i, t_f]$. The difference $t_f - t_i$ is known as the *sequence duration* – it describes how far ahead the robot is planning. The sequence duration is an important parameter, and is discussed in detail in Section 7.3.6.

Unlike the sequence of actions, the robot's trajectory η can be described quite generally by specifying the time evolution of the robot's position (x, y, θ), its linear and angular velocity (v, ω), and its linear and angular acceleration $(\dot{v}, \dot{\omega})$. We write this as

$$\eta(t) = [x, y, \theta; v, \omega; \dot{v}, \dot{\omega}] \tag{7.3}$$

Naturally, the trajectory is defined with respect to the odometric coordinate system.

Given that the actual wheel speeds match the commanded wheel speeds (to a very good approximation, at least), the trajectory η that will result from a sequence of actions ζ can be readily determined. Inspecting Figure 7.1, it is apparent that the linear and angular velocities (v, ω) are related to the wheel speeds (s_1, s_2) via the following equations:

$$\begin{aligned} v &= \frac{s_1 + s_2}{2} \\ \omega &= \frac{s_2 - s_1}{W} \end{aligned} \tag{7.4}$$

where W is the distance between the wheels. It therefore follows that the acceleration of the robot is given by:

$$\begin{aligned} \dot{v} &= \frac{dv}{dt} \\ \dot{\omega} &= \frac{d\omega}{dt} \end{aligned} \tag{7.5}$$

and that the position of the robot is given by:

$$x = \int_{t_i}^{t_f} v \cos\theta dt$$
$$y = \int_{t_i}^{t_f} v \sin\theta dt$$
$$\theta = \int_{t_i}^{t_f} \omega dt \tag{7.6}$$

Together, these equations constitute the model of the robot. Note, however, that they rely on two important assumptions: that the robot's PID controllers are accurate, and that there are no significant *latencies* in the system. Latency is an interesting problem, one that we will return to in Section 7.3.5.

7.3.2 Constraints

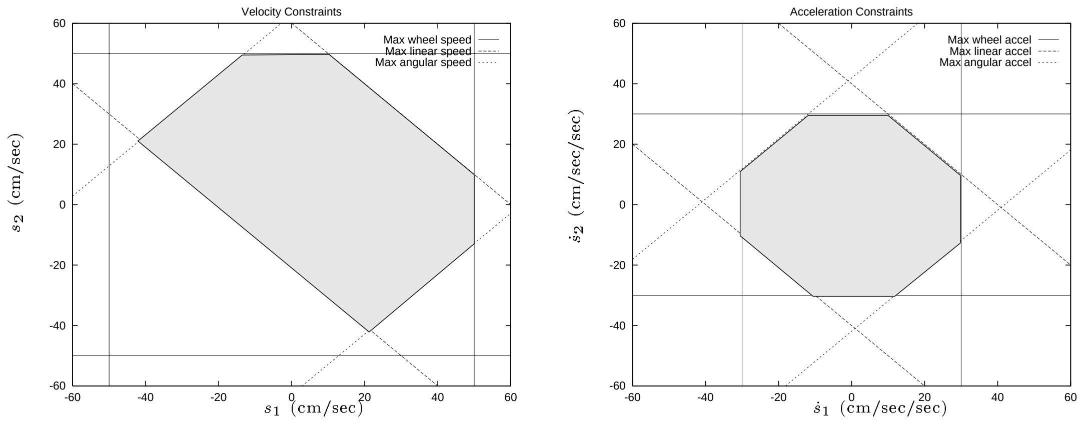
The local navigation module is equipped with a set of constraints. These constraints serve a dual role – they describe which sequences of actions are *possible*, and they describe which sequences of actions are *permissible*.

As an example of the former, consider the constraints that are imposed by the physical construction of the robot. The motors driving each wheel will have both a maximum speed and a maximum acceleration (or more accurately, a maximum torque). The robot is therefore subject to the constraints:

$$|s_{1,2}| < s_{\max}$$
$$|\dot{s}_{1,2}| < \dot{s}_{\max} \tag{7.7}$$

where $s_{\max}$ is the maximum wheel speed and $\dot{s}_{\max}$ is the maximum wheel acceleration. Any sequence of actions ζ proposed by the arbiter must, at the very least, satisfy these constraints.

As an example of the latter, consider the following constraints.



(a) (b)

Figure 7.2: (a) Constraints on robot wheel speeds (s_1, s_2). The horizontal and vertical lines indicate the direct constraints on the wheel speeds; the diagonal lines indicate the constraints on the robot's linear and angular speed. The shaded region indicates the allowed wheel speeds. (b) Constraints on robot wheel accelerations $(\dot{s}_1, \dot{s}_2)$. The horizontal and vertical lines indicate the direct constraints on the wheel accelerations; the diagonal lines indicate the constraints on the robot's linear and angular accelerations. The shaded region indicates the allowed wheel accelerations.

- The robot's linear velocity is limited to ensure it stays within safe limits (we don't want the robot moving so fast that it damages itself in the event of a collision).
- The robot's angular velocity is limited to minimise motion blur in the camera.
- The robot's linear acceleration is limited to make sure the robot doesn't topple over when starting or stopping.
- The robot's angular acceleration is limited to ensure that the torques experienced by the robot are within structural limits (in practice, this is only a concern if parts of the robot are not properly secured).

The robot is therefore subject to the constraints:

$$\begin{aligned} v_{\min} \leq v &\leq v_{\max} \\ |\omega| &\leq \omega_{\max} \\ |\dot{v}| &\leq \dot{v}_{\max} \\ |\dot{\omega}| &\leq \dot{\omega}_{\max} \end{aligned} \tag{7.8}$$

Note that these constraints are expressed in terms of the robot's trajectory η, rather than the sequence of actions ζ. We can, however, convert them into constraints on the sequence of actions using the model of the robot described in Equations 7.4 to 7.6:

$$\begin{aligned} 2v_{\min} \leq s_1 + s_2 &\leq 2v_{\max} \\ |s_2 - s_2| &\leq W\omega_{\max} \\ |\dot{s_1} + \dot{s_2}| &\leq 2\dot{v}_{\max} \\ |\dot{s_2} - \dot{s_1}| &\leq W\dot{\omega}_{\max} \end{aligned} \tag{7.9}$$

These constraints are illustrated in graphically in Figure 7.2, which shows the range of allowed actions in both (s_1, s_2) and $(\dot{s}_1, \dot{s}_2)$ space.

Note that all of the above constraints can be expressed as *behaviours*. The maximum-speed constraint, for example, can be framed as a velocity-dependent behaviour that assigns very low utility to very high velocities. However, a key difference between behaviours and constraints is that while the former are generally time-dependent, the latter are always time-*independent*. Consequently, constraints can be used to reduce the size of the arbiter's search space. This, in turn, increases the probability that the arbiter will find a near-optimal sequence. In practice, this is done by 'pre-compiling' the constraints to form a *constrained action graph*.[2] This is a graph in which nodes correspond to velocities that satisfy the velocity constraints, and arcs correspond to accelerations which satisfy the acceleration constraints. Consequently, any path through the constrained action graph is a valid sequence of actions. See, for example Figure 7.3, which shows a fragment of the constrained action graph for J Edgar.

The algorithm used to construct the constrained action graph is as follows.

- The wheel velocity space (s_1, s_2) is discretized and a node is assigned to each point in this discretized space. Nodes that violate the velocity constraints are then eliminated.

- Every node is connected to every other node via an arc. Arcs that violate the acceleration constraints are then eliminated.

[2]If constraints are *holonomic*, we can reduce the dimensionality of the search space by defining a new set of generalised coordinates [101]. Unfortunately, any constraint that is expressed in terms of an inequality is necessarily non-holonomic. More subtle techniques, such as the constrained action graph, must therefore be used.

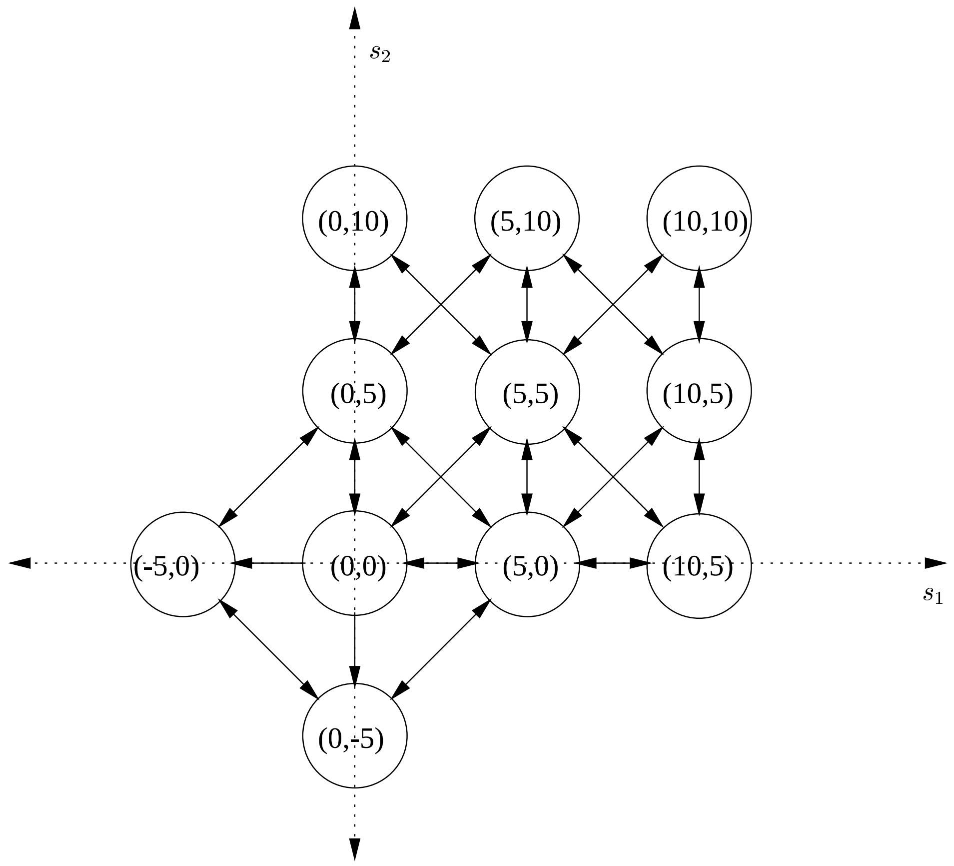


Figure 7.3: A fragment of the constrained action graph. Each node corresponds to one possible set of wheels speeds (the node marked $(0, 5)$, for example, indicates that the left wheel is stationary and the right wheel has a speed of 5 cm/sec). The arcs indicate allowed changes in speed.

Note that, in order to generate the arcs in this graph, the sequence of actions must be discretized – i.e. it is assumed that a new action is selected every Δt seconds. In practice, this is exactly what happens – the robot is sent a new velocity command at periodic intervals, typically on the order of 100ms.

7.3.3 Behaviours

In addition to the set of constraints, the local navigation module is equipped with a set of behaviours whose role is to determine the *utility* of different sequences of actions. Mathematically, behaviours are expressed in terms of a *utility function* $U(\zeta, \eta)$ which assigns a utility to a sequence of actions ζ and the corresponding trajectory η. The *total* utility of a sequence ζ is given by a weighted sum of the utility functions for each individual behaviour:

$$U_{\text{total}}(\zeta, \eta) = \sum_i k_i U_i(\zeta, \eta) \tag{7.10}$$

where the set of terms $\{k_i\}$ are known as *gain* terms. The overall behaviour manifested by the robot will depend upon the relative strength of the various gain terms. Consider, for example, a robot that is equipped with two behaviours: avoid-obstacle and move-to-goal. If the gain on the avoid-obstacle behaviour is very high (relative to the gain on the move-to-goal behaviour), the robot will tend to take very long paths around obstacles, possibly increasing the amount of time required to reach the goal. If, on the other hand, the gain on the avoid-obstacle behaviour is set very low, the robot will tend to 'cut-corners' on its way to the goal – i.e. it will follow paths that only just avoid collisions. Clearly, these gain terms must be set according the task being attempted – they may even vary over time as the task changes. In practice, some degree of experimentation is involved in setting these terms. Note that since the $U(\zeta, \eta)$ terms are not normalised, and can take any value in the range $-\infty$ to $+\infty$, the gain k_i terms have no other physical or mathematical interpretation – they simply dictate the way in which individual behaviours are combined to produce an overall behaviour for the robot.

The local navigation module implements four different behaviours: avoid-obstacle, move-to-goal, move-forward and avoid-wheel-slip.

Avoid-Obstacle

To prevent the robot from colliding with obstacles, an 'avoid-obstacle' behaviour is used. This behaviour assigns a high negative utility to sequences that take the robot towards obstacles. It has the following utility function:

$$U(\zeta, \eta) = -\int_{t_i}^{t_f} [\text{Prox}(x, y) - R]^{-1} dt \tag{7.11}$$

where $\text{Prox}(x, y)$ measures the distance between the robot and the nearest obstacle, and R is the radius of the robot. The distance $\text{Prox}(x, y)$ is determined using the local occupancy map. This function assigns a utility of $-\infty$ to sequences in which the robot collides with an obstacle. In addition, it encourages the robot to seek open spaces by assigning a high utility to sequences in which the robot spends most of its time well away from obstacles.

Move-to-Goal

The move-to-goal behaviour encourages the robot to move towards a local goal. This goal can be specified either as a 2D position, or as a 3D position and orientation. The latter is particularly useful for docking tasks. The move-to-goal behaviour assigns a high utility to sequences that take the robot closer to the goal, and a low utility to sequences that take it farther from the goal. It has the following utility function:

$$\begin{aligned} U(\zeta, \eta) &= + \left([x(t_i) - x_{\text{goal}}]^2 + [y(t_i) - y_{\text{goal}}]^2 \right)^{\frac{1}{2}} \\ &\quad - \left([x(t_f) - x_{\text{goal}}]^2 + [y(t_f) - y_{\text{goal}}]^2 \right)^{\frac{1}{2}} \\ &\quad - |\omega(t_f)| \end{aligned} \tag{7.12}$$

where $(x_{\text{goal}}, y_{\text{goal}})$ is the goal position. This function simply computes the difference between the distance to the goal at the start of the sequence, and the distance to the goal at the end of the sequence. That is, it determines *how much closer* the robot will get to the goal as a result of a particular sequence of actions.

The utility function also has a velocity dependent term $\omega(t_f)$ whose purpose is not immediately obvious. This term is used to encourage the robot not only to reach the goal, but to *stop* when it reaches the goal. The need for an explicit velocity dependant term was discovered after early experiments yielded some unexpected behaviour – the robot would move to the goal, but instead of stopping, it would either turn on the spot, or execute very tight orbits around the goal. After some consideration, it was realised that without an explicit velocity dependent term, stopping at the goal was no more desirable than turning on the spot, and was only slightly more desirable than executing orbits. The $\omega(t_f)$ term has the effect of encouraging the robot to stop at the goal, and the side-effect of discouraging rapid turning (if necessary, this side-effect can be eliminated by making the term dependent on the distance to the goal). As with potential-field-based techniques, the dynamic utility fusion approach occasionally suffers from the problem that the *emergent* behaviour of the system is not always that which the designer intended.

Note that, unlike the utility function for the avoid-obstacle behaviour, the utility function for the move-to-goal behaviour is not *path-dependent*. That is, we do not care what path the robot follows to reach the goal, so long as it gets there *in the end*. Thus, the robot may temporarily move away from the goal – to avoid an obstacle, for example – without being penalised. This is an important feature of the dynamic utility fusion approach: it rarely suffers from the local minima problems that beset potential-field-based techniques.

Move-forward

The move-forward behaviour encourages the robot to move *forwards*, as opposed to *backwards*. Without this behaviour, robot has no a priori preference for either direction. Most of the time, moving backwards is an undesirable behaviour (since the robot has forward-facing sensors, it will tend to back into things it cannot see). Sometimes, however, the robot *needs* to move backwards – to back out of a corner or away from an obstacle, for example. Consequently, while the move-forward behaviour assigns a low utility to negative velocities, it does not prohibit them entirely. The utility function for the move-forward behaviour has the following form:

$$U(\zeta, \eta) = -\int_{t_i}^{t_f} h(v)dt \tag{7.13}$$

where $h(v)$ is a step function such that

$$h(v) = \begin{cases} 0 & \text{if} \qquad v(t) >= 0 \\ 1 & \text{otherwise} \end{cases} . \tag{7.14}$$

This function assigns sequences a negative utility which is proportional to the amount of time spent moving backwards.

Avoid-wheelslip

Most wheeled robots use some kind of odometry to derive an estimate of their velocity and position. Errors in this process derive in part from 'slippage' between the robot's wheels and the surface over which the robot is travelling. The avoid-wheelslip behaviour helps to minimise this source of error by assigning low utilities to sequences that subject the robot's wheels to high accelerations. The utility function for the avoid-wheelslip behaviour is as follows:

$$U(\zeta, \eta) = -\int_{t_i}^{t_f} (|\dot{s_1}| + |\dot{s_2}|) dt. \tag{7.15}$$

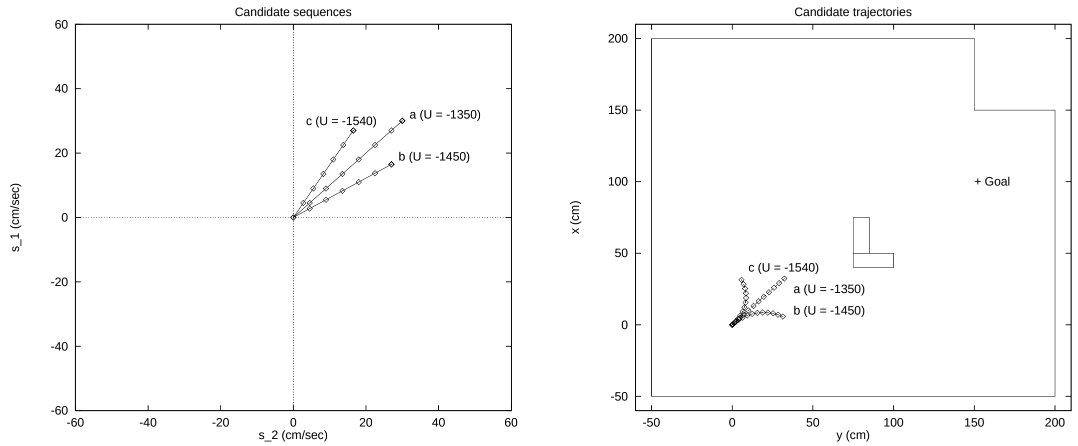
This function returns a negative utility which is proportional to the average wheel acceleration over the duration of the sequence.

7.3.4 The Arbiter

The arbiter is the key component in the local navigation module. Every 100ms or so, the arbiter must select a new sequence of actions for the robot to perform. Ideally, such sequences should be selected by considering the space of all possible sequences (i.e. those that satisfy the set of constraints) and selecting the one with the highest utility (according to the set of behaviours). See, for example, the three trajectories shown in Figure 7.4. Unfortunately, in practice, the space of all such sequences is very large, and the arbiter has very little time in which to make a decision. Consequently, the arbiter makes use of a *stochastic* search process, in which only a subset of possible sequences – selected at random – is considered.

The algorithm used by the arbiter is as follows.

- Generate a set of random sequences using the constrained action graph.



(a) (b)

Figure 7.4: (a) Three possible sequences (in wheel-speed space). The utility of each sequence is shown in brackets. (b) The corresponding robot trajectories (in position space).

- Determine the total utility of each of the sequences in this set.
- Execute the sequence with the highest utility.

Naturally, the number of sequences generated and tested depends upon the available time. In practice, therefore, the arbiter keeps generating and testing sequences until it runs out of time, at which point it uses whatever sequence has the highest utility.

Recall that sequences are constructed by selecting a random path through the constrained action graph – the arbiter simply chooses arcs at random, starting from the node corresponding to the current wheel speeds. Since there are only a finite number of sequences that can be generated in this way, the total size of the arbiter's search space can be (approximately) determined. Let n denote the number of *discrete* steps in a sequence of actions; n is determined by the sequence duration $t_f - t_i$ and action duration Δt:

$$n = \frac{t_f - t_i}{\Delta t} \tag{7.16}$$

Let α denote the average number of arcs possessed by each node (i.e. the average *connectivity* of the constrained action graph); the value of α will depend upon the nature of the constraints in the constraint set – the more heavily constrained the problem is, the lower α will be. Let N denote the average number of sequences that start from an given node; clearly, N must be such that:

$$N = \alpha^n \tag{7.17}$$

Typical values for α and n are 10 and 15 respectively, yielding 10^{15} possible sequences.

Using the stochastic search algorithm described above, the arbiter can typically generate and test at most 1000 sequences in the time available to it. That is, it will explore less than 0.0000000001% of the search space. The question must therefore be asked – why does this procedure work at all? The answer lies in the fact that while it is very difficult to find the *optimal* solution, it is relatively easy to find a *near optimal* solution. To put this another way, the arbiter will almost never find the best sequence of actions, but will almost always find a sequence that is *good enough*. This effect is quantified in Section 7.4, where we look at some experimental results.

There is one exception to the procedure described above: the first sequence the arbiter generates will *always* correspond to the *maximal braking* sequence. That is, the arbiter constructs a sequence that will bring the robot to a complete stop in the shortest possible time, subject to the acceleration constraints. In this way, the arbiter can guarantee that the robot will avoid collision with any obstacle that has been detected by the robot's sensors.

In this section, it has been assumed that the arbiter selects a new sequence of actions every 100ms, i.e. it operates at a rate of 10Hz. In principle, the arbiter might operate better at a lower rate – at 1Hz, for example – since this would give it time to explore a larger portion of the search space. Unfortunately, at a rate of 1Hz, the robot would respond very slowly to changes in its environment – the time between detecting an obstacle and starting to swerve around it, for example, could be as long as 2 seconds. At 10Hz, however, the robot will respond tolerably well to the presence of moving obstacles, such as people and other robots. Unfortunately, neither J Edgar nor Tigger will support higher rates than this: 10Hz is the maximum rate at which the robot hardware will accept new commands.

7.3.5 Latency

As noted in Section 7.3.1, an important aspect of the dynamic utility fusion approach is the ability to predict the outcome of actions, i.e. to determine the trajectory that will result from a particular sequence of actions. An important element of this prediction process is to take into account any *latencies* in the system. Consider, for example, the time that elapses between the *evaluation* and *execution* of a sequence. If the arbiter is operating at 10Hz, this delay could be up to 100ms. If the robot is travelling at high speed, its position may have changed significantly in this time. The arbiter must take this change into account when predicting the trajectory generated by a sequence of actions. Fortunately, this can be done fairly easily – the arbiter can use the model described in Equations 7.4, 7.5 and 7.6 to predict where the robot *will be* when the sequence is executed, and generate a trajectory based on this prediction.

Latencies are also present in the robot's communication links and in the PID controllers. It takes a finite amount of time to transmit a new command to the robot (from a few milliseconds to a few tens of milliseconds), and it takes a finite amount of time for the PID controllers to match the commanded wheel speeds (typically a few tens of milliseconds). The local navigation module makes no attempt to model these latencies, since they only become significant at speeds in excess of 100 cm/sec.

7.3.6 Sequence Duration

The sequence duration is an important parameter for the local navigation module. The question must therefore be asked – how long should sequences be? There are a number of advantages associated with long sequences.

- Oscillations are suppressed. Oscillations occur when the robot rapidly switches back-and-forth between two actions. Consider, for example, a robot placed between a pair of parallel walls. If the sequence duration is very short, the robot will tend to 'bounce' back and forth between the walls (this is the sort of behaviour often observed in potential field approaches). If the sequence duration is longer, the robot will follow a path that is parallel to, but equidistant from, both walls.

- The probability that the robot will get stuck in some local minimum is reduced. Consider, for example, a robot which has an obstacle between it and the goal. If the sequence duration is very short, the robot may be unable to find any sequence that takes it around the obstacle – it will become stuck behind the obstacle. With a longer sequence, it will move around the obstacle.

On the other hand, there are also a number of disadvantages associated with long sequences.

- The size of the search space increases dramatically. Since the arbiter has only a fixed amount of time in which to make decisions, it will have less chance of finding a near-optimal trajectory.

- The predicted trajectories become less reliable. It is relatively easy to predict the state of the robot 100ms in the future; it is much more difficult to reliably predict the state of the robot 10s in the future. This is partly due to the uncertainty in the outcome of individual actions and partly due to the fact that the environment itself may be changing.

Clearly, there will exist a 'sweet-spot' that lies between the two extremes of very long and very short sequences. The location of the sweet-spot is dictated by the physical properties of the robot, the behaviours being used and the environment the robot inhabits. The ideal sequence duration must therefore be determined experimentally. For J Edgar and Tigger, sequence durations of between one and two seconds have been found to be most appropriate.

It is possible, however, to place a *lower bound* on the sequence duration by considering the constraints on the robot's motion. For example, if the robot is to avoid collisions with stationary obstacles, the sequence duration must be long enough to allow the robot to come to a complete stop. That is, the sequence duration must be such that

$$t_f - t_i > v_{\max}/\dot{v}_{\max}. \tag{7.18}$$

Similarly, if the robot is to reach a goal that is directly behind it, the sequence duration must be long enough to allow the robot to turn through 180°. That is, the sequence duration must be such that

$$t_f - t_i > \omega_{\max}/\dot{\omega}_{\max}. \tag{7.19}$$

For J Edgar and Tigger, these two constraints imply that sequences must have a duration of at least 1 second.

Note that, to some extent, it is possible to make a trade-off between the length of sequences and the complexity of behaviours. For example, rather than expecting the arbiter to discover that it must decelerate when nearing an obstacle, one can express this requirement through the avoid-obstacle behaviour itself. Using the utility function, one can say that when the robot is near an obstacle, high velocities have a low utility. Thus, the robot would tend to decelerate when approaching an obstacle, even if it is not looking far enough ahead to realise that a collision is imminent. Effectively, by encoding more knowledge into behaviours, we can make it easier for the arbiter to find near-optimal solution. The extent to which this trade-off *should* be made remains an open question.

7.4 Experiments

Three different experiments are described in this section. The first two experiments, using the robots Tigger and J Edgar, demonstrate the basic functionality of the local navigation module. In particular, these experiments illustrate the goal seeking and obstacle avoidance behaviours. The third experiment, which was conducted using a simulator, establishes the relationship between the number of sequences that are tested, and the utility of the best sequence found by the arbiter. This experiment demonstrates that, for most situations, it is not very difficult to find a near-optimal sequence.

Note that, while these experiments are far from exhaustive (they do not, for example, examine the effects of varying the sequence duration), they do suffice to demonstrate the effectiveness of the local navigation module.

7.4.1 Tigger

Figure 7.5(a) shows the environment used to evaluate the local navigation module on the robot Tigger. This environment was constructed by laying carpet tiles on a linoleum floor – the robot sees the edges of the tiled area as obstacles. A box was also placed on the floor to act as an obstacle. For this experiment, the robot was placed at the bottom-left hand corner of the environment and given a goal at the top right corner; all *global* navigation was disabled.

Figure 7.5 also shows the results for this experiment: Figure 7.5(b) shows the

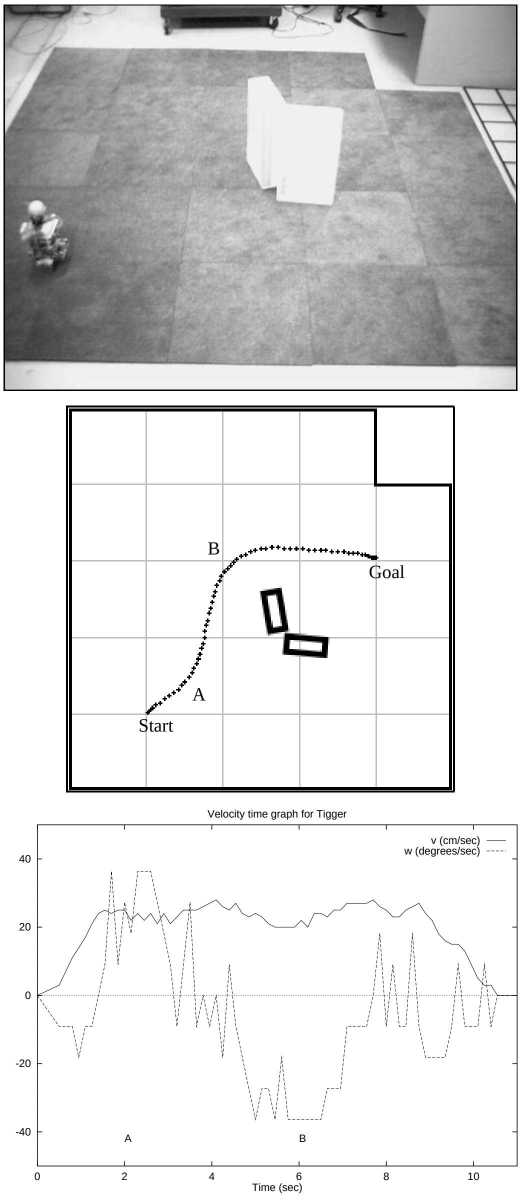


Figure 7.5: Results for the experiment with robot Tigger. (a) Photo of the experimental setup. (b) Map showing the robot's path (as determined using odometry). (c) Graph showing the robot's velocity as a function of time.

path taken by the robot (according to odometry) and Figure 7.5(c) shows the robot's velocity as a function of time. There are two important features that are apparent in these results:

- **Smooth cornering.** Upon detecting the obstacle (point A in the figure), the robot executes a smooth left turn; when it is clear of the obstacle (point B in the Figure), the robot executes a smooth right turn. Looking at the velocity-time graph, we can see that the robot's speed drops slightly during these turns, as it first decelerates into the turn and then accelerates out again. Note that the change in speed is relatively small, as the radius of curvature of the turn is quite large.

- **Smooth acceleration**. From its starting position, the robot quickly accelerates up to its maximum speed of 30 cm/sec. It slows somewhat to swerve around the obstacle (points A and B), then decelerates to a stop at the goal. At no point does the robot undergo sudden accelerations, and the average speed over the course of the experiment is high.

Note that the velocity-time graph for this experiment is quite 'noisy'. This noise arises mostly from uncertainty in the odometric velocity estimates (encoders have limited resolution and integration time is short), and partly from the random nature of the search algorithm used by the arbiter. Since the arbiter is selecting a new sequence of actions every 100ms, the robot may rapidly switch between different velocities. At no point, however, is the change in velocity such that the acceleration constraints are violated. Consequently, despite the noise, the path followed by the robot (in position space) is very smooth.

The results shown in Figure 7.5 seem all the more remarkable when one considers that the average number of sequences tested by the arbiter in each 100ms interval was less than 50. That is, the arbiter was able to find good sequences of actions after testing only a tiny fraction of the space of all possible sequences. Clearly, for this situation at least, finding near-optimal sequences is not very difficult.

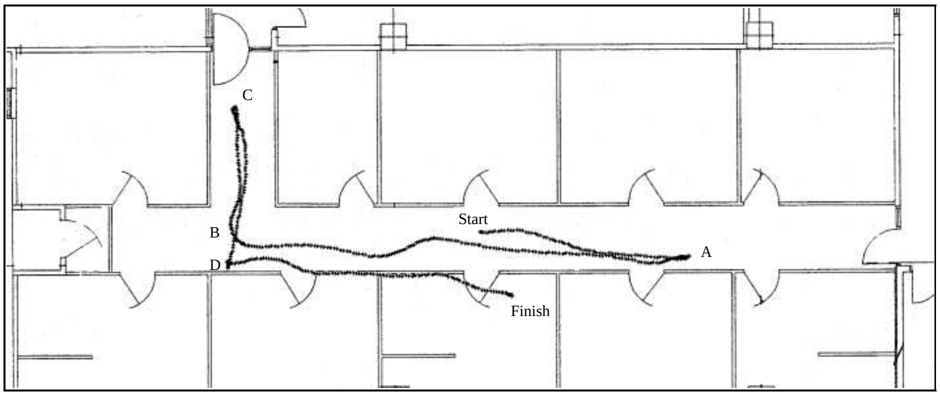
7.4.2 J Edgar

Figure 7.6(a) shows the environment used to evaluate the local navigation module with robot J Edgar. This is a section of office corridor. In this experiment, the robot was 'joysticked' around the environment – i.e. joystick commands were fed into the local navigation module as a local goal. All global navigation was disabled.

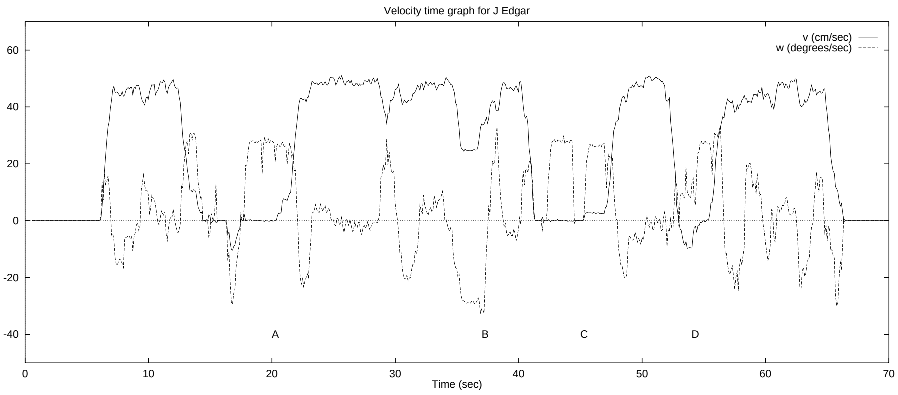
Figure 7.6 also shows the results for this experiment: Figure 7.6(b) shows the path taken by the robot (according to odometry), Figure 7.6(c) shows the robot's velocity as a function of time. In reality, the robot was taken on a tour that returned it to its starting position. Some degree of odometric drift, however, is apparent in Figure 7.6(b). There are a number of events worth noting in these results.

- At points A and C, the robot performed in-place turns in response to sudden changes in the goal position – the robot decelerated to a stop, turned through 180 degrees and accelerateed up to speed once again.
- At point B, the robot executed a smooth turn through 90 degrees to handle a corner.
- At point D, the robot overshot the turn (as a result of operator inattention – the robot was given a goal position that would have required it to move through a wall). The robot handled this situation by coming to a rapid stop, backing up, and attempting the turn again.

(a)



(b)



(c)

Figure 7.6: Results for the experiment with robot J Edgar. (a) Photograph of the experimental location (a section of office corridor). (b) Map showing the robot's path (as determined using odometry). Note that there is a significant amount of odometric drift apparent in these results. (c) Graph showing the robot's velocity as a function of time.

Overall, the robot maintained a high average speed throughout the course of the experiment.

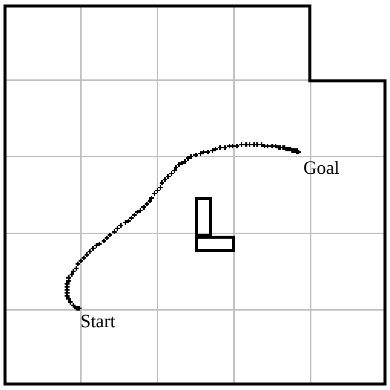
Note that, despite its size, J Edgar is capable of much higher speeds than Tigger (100cm/sec versus 50cm/sec), albeit at lower accelerations (30cm/sec/sec versus 50cm/sec/sec). For these experiments, however, the speed of the robot was limited to 50cm/sec. This was partly for the sake of safety (the robot is quite dangerous at these speeds, both to itself and to others), but mostly because of the limited field-of-view of the robot's camera. This field-of-view is such that the robot can only detect obstacles that are within 120cm; furthermore, the robot has an effective radius of 30cm. Consequently, the robot must be able to come to a complete stop in less than 90cm if it is to reliably avoid collisions. The stopping distance for a robot travelling at a speed of v_{max} with a maximum rate of deceleration of $\dot{v}_{\mathrm{max}}$ is given by:

$$d_{\mathrm{stop}} = \frac{v_{\mathrm{max}}^2}{2\dot{v}_{\mathrm{max}}}. \tag{7.20}$$

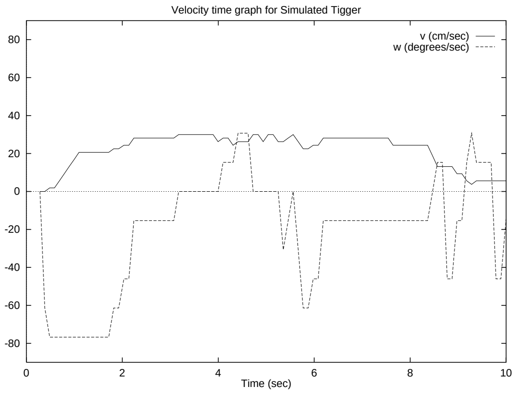
Therefore, in principle, the J Edgar can travel safely at speeds of up to 73cm/sec. In practice, once the various latencies in the system are taken into account, 50cm/sec is the maximum realistic speed. Higher speeds would require either higher accelerations or sensors with a longer range.

7.4.3 Simulation

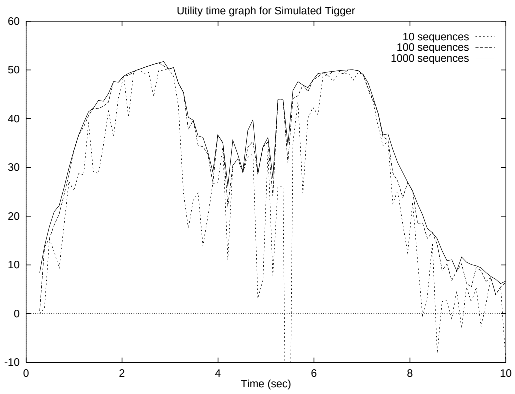
Figure 7.7(a) shows the simulated environment used to assess the relationship between the number of sequences tested and the utility of the sequence selected by the arbiter. The robot was placed at the bottom-left corner of the environment and was given a goal at the top right corner. Initially, the robot was facing away from the goal.



(a)



(b)



(c)

Figure 7.7: Simulation results. (a) Map showing the path of the robot through the simulated environment. (b) Velocity time graph. (c) Graph showing the utility of the selected sequence as a function of time.

In the experiments involving Tigger and J Edgar, the arbiter tested an average of 50 sequences in each 100ms interval. In this experiment, the simulation was slowed down to allow the arbiter to test many more sequences. Figure 7.7(c) shows the results — this graph plots the utility of the selected sequence as a function of time. The three curves in this graph correspond to the utility of the best sequence found after testing 10, 100 and 1000 sequences. Inspecting this graph, it is clear that the sequences found after 100 tests are much better than those found after 10 tests. One the other hand, there is very little difference between the sequences found after 1000 tests and those found after 100 tests. Clearly, for this situation at least, it is very easy to find a near optimal sequence. It is not surprising, therefore, that the local navigation module works well even when it is testing, on average, only 100 sequences in each 100ms interval.

7.5 Conclusion

The experiments described in Section 7.4 demonstrate that the dynamic utility fusion approach described in this chapter works well – the motion produced by the local navigation module is both fast and smooth. The system has also undergone a number tests of a much more difficult (but much less formal) nature. During University Open Days, members of the public (principally children) are given the opportunity to 'joystick' one or more of the robots around a specially laid out environment (the joystick commands are fed into the local navigation module as a nearby goal). Very few collisions have been observed on these occasions, which is surprising when one considers that many of the children are *trying* to crash the robot. Furthermore, these collisions appear to arise from a failure of the local mapping module, rather than the local navigation module per-se. Typically, the robot turns into a nearby obstacle that was just outside its field-of-view. Further testing under controlled conditions needs to be done in order to properly assess the failure modes of the local navigation module.

There are many variations to the basic DUF approach that could be pursued; I will mention just two here.

The first variation involves extending the prediction process to handle *uncertainty* in the outcome of actions. That is, the model of the robot can be extended to reflect the fact that any given sequence of actions may give rise to many *different* trajectories, each of which has a different probability. With this scheme, the total utility for a sequence of actions ζ is given by

$$U(\zeta) = \sum_{\eta} U(\zeta, \eta) p(\eta \mid \zeta) \tag{7.21}$$

where $p(\eta \mid \zeta)$ is the probability that the robot will follow the trajectory η, given that it executes the series of actions ζ. For robots operating in indoor environments, this extension is probably unnecessary (over the timescales we are considering, the motion of the robot is highly predictable). In outdoor environments, where the terrain is rougher, this extension may well be worth considering (indeed, such an extension can already be found in Rosenblatt's DAMN architecture [204, 203]).

The second variation involves extending the arbiter so that it is capable of *learning* which sequences of actions are more useful. That is, instead of simply testing randomly generated sequences, the arbiter could keep track of those sequences that have worked well in the past, and apply these sequences to current problems. The difficulty with this particular extension is that it is far from clear how sequences can be encoded to facilitate rapid learning. This particular topic is the subject of ongoing research.

Chapter 8

Landmark Detection

> To see a thing one has to comprehend it. An armchair presupposes the human body, its joints and limbs; a pair of scissors, the act of cutting. What can be said of a lamp or a car? The savage cannot comprehend the missionary's Bible; the passenger does not see the same rigging as the sailors. If we really saw the world, maybe we would understand it.

from *The Book of Sand*, by Jorge Luis Borges.

Abstract

This chapter describes a mechanism for detecting landmarks – such as walls, doorways and intersections – using the local occupancy map. These landmarks are used by the global localisation module to help determine the robot's location.

8.1 Introduction

Determining the location of a robot with respect to the environment is an important aspect of mobile robot navigation. There are four basic methods the robot can employ for this task:

- The robot can use a dead-reckoning system, such as odometry or inertial navigation. However, since dead-reckoning systems are prone to cumulative drift, they are, by themselves, inadequate for the task of localisation.

- The robot can use a global positioning system, such as satellite GPS. Unfortunately, GPS cannot be used in environments such as office buildings, factories and underground mines.

- The robot can make use of pre-placed artificial beacons, such as bar-coded rods or coloured tags. Not all environments, however, can be fitted with artificial beacons.

- The robot can make use *naturally occurring landmarks*; ideally, these landmarks will be *time-invariant* features of the environment.[1]

It is this last of these methods that is employed in this thesis. Consequently, this chapter describes a method for identifying natural landmarks – such as walls, doorways and intersections – in indoor environments. In Chapter 9, I will show how these landmarks can be used to localise the robot with respect to a *model* of the environment.

The Landmark Detection Module

The *landmark detection module* that is responsible for detecting significant landmarks. Instead of looking for landmarks in the robot's raw sensor data (in camera images, for example), this module looks for landmarks in the *local occupancy map*. There are a number of good reasons for doing this.

- Key landmarks, such as doorways and intersections, are fairly abstract features. Detecting these landmarks in raw sensor data can be extremely difficult, while detecting the same landmarks in the local occupancy map is a relatively straight-forward process.

- Landmarks that appear only briefly in the raw sensor data will persist for some time in the local occupancy map. It may take several seconds, for example, for the robot's motion to carry a landmark beyond the bounds of the local occupancy map. Consequently, the landmark detection process can be relatively slow.

[1] Note that, since the environment is assumed to be static throughout this thesis, the time-invariance of landmarks is not an really an issue.

- The same landmark detection module can be used with a number of different sensors, so long as those sensors can be used to produce a local occupancy map. In this thesis, we have considered occupancy maps derived from sonar and vision, but one could equally well use laser range finders or radar.

It is still possible, of course, to design a module for detecting landmarks in the raw sensor data. One could, for example, look for extended line segments in the raw camera images. However, for the reasons stated above, the navigation system described in this thesis uses landmarks extracted from the local occupancy map.

To locate landmarks, the landmark detection module makes use of a simple template matching scheme. The module maintains a relatively large set of templates, each of which corresponds to a particular landmark at a particular orientation. These templates are used to search the local occupancy map for matching landmarks. The key challenge for the landmark detection module lies in making this process fast enough to be useful. A typical set of templates, for example, will contain 128 elements (corresponding to 4 different types of landmarks at 32 different orientations), while a typical map will contain 6400 different locations at which these templates might be tested (corresponding to a local map that is 4m on a side, with 5cm cells). This yields a total of 819,200 possible comparisons. Clearly, a very fast matching mechanism is required. In Section 8.3, I will describe one such mechanism, which is based on the notion of 'early jump out' [56]. Put simply, when searching the local map, the landmark detection module will quickly abandon comparisons that are *unlikely* to lead to good matches. In fact, as I will show in Section 8.4, the vast majority of potential matches can be eliminated after inspecting only one or two map cells. As a result, use of this early-jump-out mechanism can lead to a 10-fold improvement in the overall speed of the system.

Chapter Structure

The remainder of this Chapter is structured as follows. Section 8.2 reviews some related work on landmark detection for mobile robot applications. Section 8.3 describes, in detail, the landmark detection module, while Section 8.4 evaluates the performance of this module in an experimental context. Section 8.5 draws some conclusions and indicates a range of possible enhancements to the somewhat simplistic approach described in this chapter.

8.2 Related Work

In this section, I will consider some of the different types of landmarks that are used for mobile robot localisation. In doing so, I will take a somewhat liberal interpretation of what constitutes a landmark (the term *landmark* is readily understood but difficult to define). Landmarks can be classified according to a number of different criteria:

- The sensor modality from which the landmarks are extracted – sonar, vision, laser range finders, or some intermediate representation (such as occupancy grids).
- Whether the landmark is used in indoor or outdoor environments.
- Whether the landmark is natural or artificial. In this context, a *natural* landmark is a pre-existing feature of the environment, irrespective of whether or not that environment is natural or built. In contrast, an *artificial* landmark is some object that is added to the environment specifically for the purpose of aiding robot localisation.

- Whether the landmark is active or passive. This applies only to artificial landmarks, of course.

- Whether the landmark is a pre-defined feature selected by a human operator, or is selected automatically by the robot. This applies only to natural landmarks.

This review will concentrate mainly on indoor environments, and on landmarks derived from visual information or from occupancy grids. The review is therefore divided into four sections:

- Detecting artificial landmarks using vision.

- Detecting natural landmarks using vision.

- Detecting natural landmarks using occupancy grids.

- Automatic selection of natural landmarks.

Note that the first and last sections are included more for the sake of completeness than for their direct relevance to this chapter.

Detecting Artificial Landmarks using Vision

Fitting the environment with *artificial* landmarks, in the form of easily identifiable visual patterns, is pragmatic solution to the localisation problem. Patterns that have been used for this purpose include: black squares on a white background [141]; alternating black and white strips [135]; black circles on a white background [10]; red and white road cones [207]; and patterns of coloured lights [48]. Typically, these landmarks are mounted high on the walls or ceiling to maximise their visibility.

However, a limiting factor with simple patterns such as these is that all of the landmarks are *identical*; this introduces a correspondence problem into the localisation process. This correspondence problem can be avoided only if each landmark in the environment is uniquely identifiable. Hence, a number of authors have employed patterns that, in addition to being easily detectable, encode their unique identity. Examples include Kortenkamp *et al* [145], who employ binary-coded black-and-white rods, and Ellepola *et al* [80, 79], who employ binary-coded black-and-white disks.

In some ways, the use of artificial landmarks makes the localisation problem *too* easy – many of the navigation problems discussed in Chapter 9 become moot if the environment is littered with uniquely identifiable landmarks. Hence, this thesis is concerned with the more difficult problem of navigation using *natural* landmarks.

Detecting Natural Landmarks using Vision

The vast majority of the vision-based natural landmark detection systems that have been described to date make use of either point-like or line-like image features as landmarks [12, 14, 36, 62, 88, 106, 148, 154, 170, 253, 262, 266]. Systems based on *vertical* lines are particularly popular. There are two arguments generally put forward for using vertical lines. Firstly, long vertical lines generally correspond to *invariant* features of the environment, such as windows, doorways, the sides of bookshelves, and so on. Secondly, for robots using stereo vision, the range of vertical lines can be determined very accurately, and very robustly; in contrast, horizontal lines yield much less accurate range information. (With vertical lines, one need only know the line's position on the image's x-axis; with horizontal lines, one must rely on the matching the end-points of the lines, whose location is generally not robust to errors in the low-level edge extraction process).

Another popular approach is to use features on the ceiling as landmarks [1, 2, 70, 71, 96]. Once again, the argument for using these features is that they are largely invariant. In addition, these landmarks have the advantage that they are rarely obscured and remain visible over the full range of robot orientations. Robots employing such landmarks are generally equipped with a camera that points straight up at the ceiling. Dulimarta *et al* [70, 71] go even futher, describing a robot with two cameras: one pointing straight up, to observe features on the ceiling, and the other pointed straight out, to read the numbers off doors.

As an alternative to using image features as landmarks, a number of authors have considered the possibility of detecting landmarks through *direct* image matching. In this scenario, the robot operates in two different modes. During a *recording* run, a human operator guides the robot through the environment while the robot records a series of images. During an *autonomous* run, the robot determines its location by matching the observed images against those learnt during the recording run. The image matching might be done using simple correlation [109, 125], or it might involve more complex methods, such as the use of Fourier analysis [164, 165, 257] or image eigenspaces [162]. These techniques are simplified somewhat by the use of an omni-directional camera, which gives the robot a 360° field of view.

While all of the approaches described above have their merits, I have chosen, for the reasons stated in the Introduction, not to employ any of them. There are, in addition, certain technical considerations: the robots I am using are not equipped with multiple cameras, and the one camera the robots do have is pointed at the floor (to detect obstacles). To detect vertical lines or features on the ceiling, this camera would have to be tilted back, which would mean that the camera would have to alternate between obstacle detection and landmark detection tasks. While this is an interesting problem in itself, it is a complication I wanted to avoid. In particular, I wanted to avoid a situation where the robot has to periodically stop and 'take a look' to see where it is.

Detecting Natural Landmarks using Occupancy Grids

A number of authors have considered the problem of extracting natural landmarks from occupancy grids. The methods used mirror those used to extract landmarks from raw image data – one can either look for a *direct* match with a previously learnt grid, or one can attempt to extract *features* from the grid.

Both Thrun [243] and Yamauchi [258] describe methods for matching a local occupancy grid against a previously learnt (global) grid. The basic problem is to find a translation and orientation of the local grid such that it matches the previously learnt grid. Both authors use gradient descent to speed up this search process. One difficulty with these techniques is that the match functions for grid-based representations tend to be non-differentiable, and hence not well suited to gradient descent search. To overcome this difficulty, Thrun uses *interpolation* to produce a match function which *is* differentiable; this both speeds up the search process and improves the accuracy of the match. Unfortunately, it is also assumes that the robot has a relatively good initial estimate of the correct translation and orientation of the local grid. In contrast, Yamauchi employs a non-differentiable match function and does not assume that the robot has a good initial estimate. As a result, the amount of computational power required to find a match is quite high; a figure of 20 seconds on a DecStation 3100 is quoted.

From a personal perspective, I have found gradient descent methods to be largely ineffective – in the absence of good initial estimate, the search tends to quickly get bogged down in local minima. While it is possible to construct more complex match functions, which do not produce a multitude of local minima, the extra computation required to evaluate these functions tends to negate their utility. Consequently, the approach described in this chapter, which employs a combination of stochastic search with early-jump-out, can locate landmarks *faster* than a gradient-descent-based method, despite its apparent simplicity.

As an alternative to direct matching of occupancy grids, some authors have looked at extracting *features* from occupancy grids [59, 116, 140, 215, 216, 268]. While these approaches are superficially attractive, each of them also has its problems.

Scheile *et al* [215, 216] describe a method for extracting straight lines using a simple Hough transform [63]. The reasoning here is much the same as the reasoning for using vertical lines in images: extended line segments most probably correspond to invariant features of the environment. Unfortunately, with a simple Hough-based approach, it is difficult to distinguish between walls and doorways; in the Hough space, there is no difference between continuous lines (walls) and broken lines (doorways). In an indoor environment, this distinction can be vitally important.

Horst [116] describes a general method for extracting *object boundary curves* from occupancy grids. This high-level description, which describes the boundary of objects in terms of piece-wise linear segments, can be analysed to find walls, corners and so on. While this technique appears promising, it is not clear that it can be made sufficiently robust for applications involving real (as opposed to simulated) data.

Courtney *et al* [59] borrow yet another technique from pattern recognition, and make use of *invariant moments* [17]. By treating the occupancy grid as a simple greyscale image, it is possible to form a number of statistical moments that are invariant to translation, rotation and scaling.[2] An occupancy grid can therefore be *classified* according to the set of moments it generates. Unfortunately, while invariant moments offer an interesting method for *classifying* occupancy grids, they do not really indicate the *location* of landmarks as such. Consequently, their applicability to landmark-based localisation is unclear.

Simmons and Koenig describe a method for extracting features from a local occupancy grid, as implemented on the Xavier robot [140]. This feature detector finds walls by projecting virtual 'rays' from the robot. When a ray intersects an occupied cell, the robot computes the line of best fit through the adjacent occupied cells; if the fit is good, the feature is classified as a wall. A similar method is used to find openings. One limitation of this approach, however, is that it does not generalise in an obvious way to more complex features. One is therefore limited to a very impoverished feature set (walls and openings).

Automatic Selection of Natural Landmarks

Within each of the paradigms described above, some attempts have been made at *automatically* selecting the environmental features that make the best landmarks [102, 163, 241, 242, 262, 269]. For example, Yeh *at al* [262] consider this problem in the context of lines extracted from a monocular image. They demonstrate a method for selecting, from the set of all possible lines, the subset which is *least likely* to be confused. This simplifies the landmark correlation problem, which in turn leads to

[2] Although in this case, one would probably be better off with scale *dependent* moments – unlike the normal 2D image recognition tasks to which moments are normally applied, scale is an important cue in occupancy maps.

more robust localisation. Thrun [241, 242] goes even further, describing an algorithm that automatically selects landmarks from raw sensor readings, based on the degree to which they improve the *accuracy* of the localisation task. A neural net is then trained to recognise those landmarks. This kind of approach seems promising, since it raises the possibility of constructing a robot that can not only learn a map of the environment, but can also learn the set of *landmarks* from which to construct that map. Regrettably, such topics are beyond the scope of this thesis.

8.3 The Landmark Detection Module

The landmark detection module has three principal components.

- A set of pre-defined landmark *templates* corresponding to the landmarks of interest. These templates are provided by a human operator; a typical set of templates is shown in Figure 8.1.

- A *match function*, which determines the degree to which a particular template, at a particular location, matches the local occupancy map. The match function returns its highest value when all of the occupied cells in the template correspond to occupied cells in the map, and all of the unoccupied cells in the template correspond to unoccupied cells in the map.

- A *search algorithm*, which determines which templates are evaluated, and at which locations.

In the following sections, I will consider each of these components in turn.

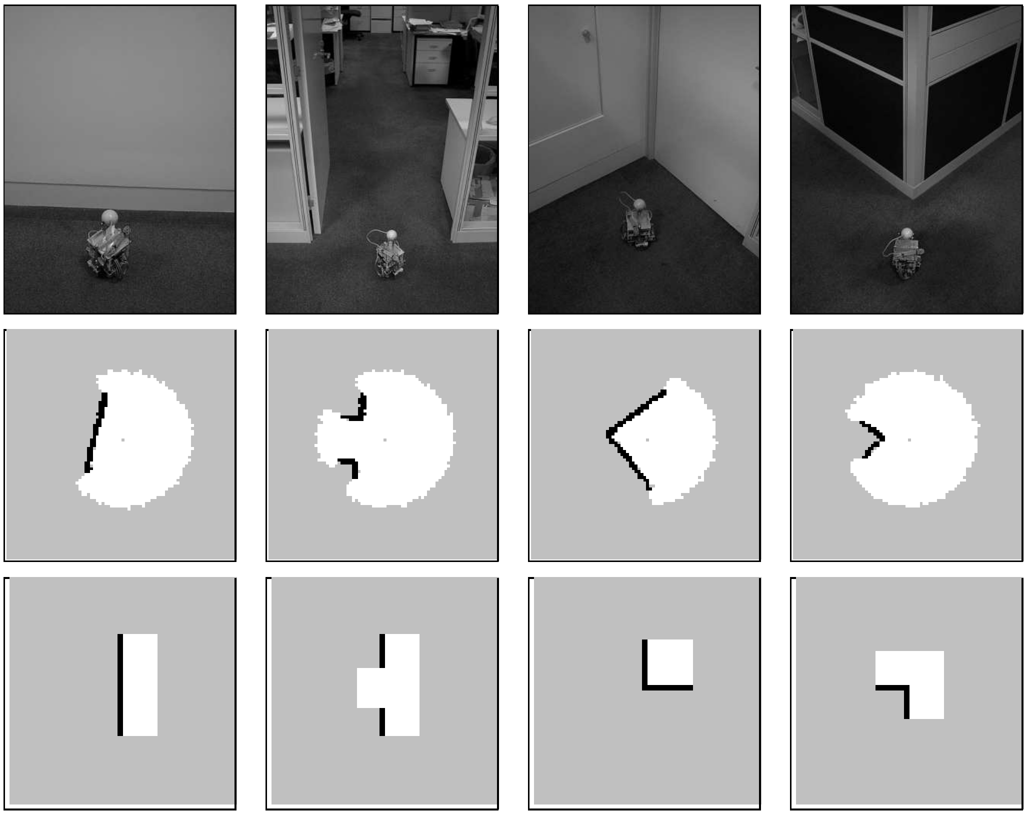


Figure 8.1: A typical set of landmarks. The top row shows the robot Tigger next to: a wall, a doorway, an inside corner, an outside corner. The middle row shows a corresponding set of local occupancy maps – dark cells in these maps indicate regions that are probably occupied, light cells indicate regions that are probably unoccupied, and grey cells indicate regions whose occupancy state is unknown. The bottom row shows the set of templates – dark cells indicate regions that should be occupied, light cells indicate regions that should not be occupied and grey cells indicate regions whose occupancy state is irrelevant.

8.3.1 Templates

These templates used by the landmark detection module are encoded as simple grids. Each cell in this grid may be in one of three states: occupied, empty, and don't care. Figure 8.1 shows a typical set of templates; the dark cells in these templates correspond to regions that should be occupied, the light cells to regions that should be empty, and the grey cells to regions whose occupancy state is irrelevant (i.e. to 'don't care' regions). To simplify the matching process, the cell size in these grids is chosen to match the cell size used in the local occupancy map.

Each of the templates used by the landmark detection module corresponds to a particular landmark at a particular *orientation*. This is done to save time during the matching process: the landmark detection module does not have to rotate templates in order to match them, instead, it can simply choose one of the variants from the set. Note that, in principle, the number of orientations required for each type of landmark depends on a number of factors, including the size of the landmark and the number of rotational symmetries it possesses. Larger landmarks generally require more orientations; landmarks that are rotationally symmetric require fewer. For the sake of simplicity, however, the landmark detection module simply uses 32 different orientations for each type of landmark.

Templates are supplied, in advance, by a human operator. The operator does not, however, have to supply templates for all possible landmarks at all possible orientations. Instead, the operator supplies the landmark detection module with a *single* template for each type of landmark. From these templates, the module automatically constructs a much larger set of templates, in which there is a template representing every possible orientation of every possible landmark.

8.3.2 The Match Function

The match function measures the similarity between a template – at a particular location – and the local occupancy map. The particular match function used by the landmark detection module simply determines the number of matching cells in the template and the map; that is, it counts the number of instances in which occupied cells in the template correspond to occupied cells in the map, and in which empty cells in the template correspond to empty cells in the map. 'Don't care' cells in the template are ignored.

The match function can be defined formally as follows. Let $\mathrm{Occ}(x, y)$ denote the local occupancy map. For every cell (x, y) in the map, this function returns one of three values

$$\mathrm{Occ}(x, y) = \{\textit{occupied}, \textit{unoccupied}, \textit{unknown}\} \tag{8.1}$$

Let $T_{ij}(\hat{x}, \hat{y})$ denote a template of type i and orientation j. For every cell $(\hat{x}, \hat{y})$ in the template, this function returns one of three values:

$$T_{ij}(\hat{x}, \hat{y}) = \{\textit{occupied}, \textit{unoccupied}, \textit{don't care}\} \tag{8.2}$$

Let the match function $S(T_{ij}, x, y)$ indicate the degree to which a template T_{ij} placed at (x, y) matches the local occupancy map. The value of this function is given by:

$$S(T_{ij}, x, y) = \frac{1}{N} \sum_{\hat{x},\hat{y}} \delta[T_{ij}(\hat{x}, \hat{y}), \mathrm{Occ}(x + \hat{x}, y + \hat{y})], \tag{8.3}$$

where the delta function returns 1 if its arguments are equal, and 0 otherwise. Note that the summation is restricted to *significant* cells in the template (i.e. to cells that are either occupied or empty), and that N denotes the total number of such cells. As a result, this expression completely ignores cells in the template that are marked as 'don't care'. On the other hand, cells in the map which are marked as 'unknown' are counted as evidence *against* a landmark being present; the match function is designed to detect landmarks that *are* present, not landmarks that *might be* present.

A template is said to match the map at location (x, y) whenever the value of the match function $S(T_{ij}, x, y)$ exceeds the pre-set match threshold $S_{\min}$. The match threshold allows for the presence of noise in the local occupancy map – if it is set too low, the landmark detection module will detect false landmarks; if it is set too high, the landmark detection module will fail to detect genuine landmarks. Typically, the match threshold is set to around 0.9, so that at least 90% of the significant cells in the template must match those in the map. The issue of selecting an appropriate match threshold is discussed further in Section 8.3.5.

8.3.3 The Search Algorithm

The choice of search algorithm for the landmark detection module is somewhat constrained by the fact that the match function defined in Equation 8.3 is non-differentiable. Consequently, as noted in 8.2, a simple gradient descent style search is not always effective. Instead, the landmark detection module employs a simple *stochastic* search. The algorithm for this is as follows:

- Step 1. Select, at random, one possible template from the set of templates.
- Step 2. Select, at random, one possible location from the set of all possible locations in the local occupancy map.
- Step 3. Evaluate the match function for this combination.
- Step 4. If the value of the match function exceeds the match threshold, notify the global localisation module that a landmark has been detected.

- Step 5. Go to Step 1.

The reason for using a random search, as opposed to a raster search, is that we do not want to introduce systematic effects that might bias the search process. For example, if the robot is in motion, we do not want the landmark detection module to spend all of its time looking for landmarks in a part of the map that is about to disappear. Another way of looking at this is to say that we want the search algorithm's coverage to be uniform with respect to the *entire* odometric coordinate system, not just the window defined by the local map. A raster search cannot do this; as the robot moves, 'gaps' in the search pattern will inevitably appear. In addition, this unbiased search process tends to produce measurements that are statistically independent, which aids the global localisation process described in Chapter 9. Note that, in the event that one is not too concerned with statistical independence, there are more sophisticated search and matching algorithms that can be used. For example, one could search only those areas of the occupancy map that have been recently updated, since this is where new matches are most likely to occur. Such extensions remain the topic of future research.

8.3.4 Early Jump Out

Since templates typically contain anywhere between 20 and 100 significant cells, computing the match function on the entire template can be quite time consuming. Therefore, to save time, the landmark detection module makes use of an 'early jump out' technique. If, during the evaluation of the match function, it appears unlikely that the match threshold will be exceeded, the match function is terminated, and returns a value of 0. As we will see in Section 8.4, this early jump out technique makes an enormous difference to the rate at which landmarks are detected.

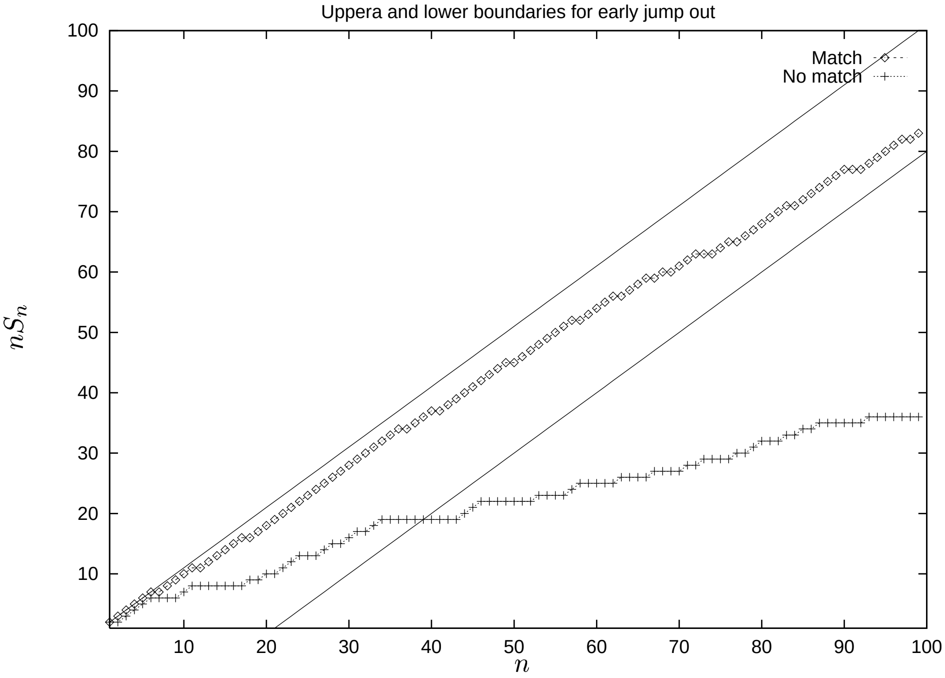


Figure 8.2: An illustration of the early jump out technique. The top curve shows the number of matching cells as a function of the number of cells tested, for a matching template. The bottom curve shows the result for a template that does not match. The lines indicate the upper and lower bounds for a matching template ($S_{\min} = 0.8N$).

The early jump out mechanism relies on the fact that the *summation* term in the match function (Equation 8.3) increases monotonically with the number of cells compared (or, strictly speaking, is monotonically non-decreasing). Alternatively, if we let S_n denote the value of the match function after n cells have been compared, the product nS_n (which simply measures the number of matching cells after n comparisons) must increase monotonically with n. Furthermore, for a *matching* template, nS_n must be such that:

$$nS_n + (N - n) \geq NS_{\min}, \tag{8.4}$$

where N is the number of significant cells in the template and $S_{\min}$ is the match threshold. For a matching template, the number of cells that have already been matched, when added to the number cells that are yet to be checked, must exceed the number of cells that need to be matched. Otherwise, even if all of the unchecked cells match, the match threshold $S_{\min}$ will not be exceeded. Figure 8.2 illustrates this situation graphically.

Equation 8.4 can be used as an early jump out *rule* – if, at any point, this equation is not satisfied, we can be certain that the template does not match, and the match function can be terminated. This early jump out leads to a significant decrease in the average time required to evaluate the match function. This time can be decreased even further if we use an early jump out rule with somewhat different properties: one that will sometimes reject true matches, as well as false ones. For example, we might decide that if the first 4 comparisons yield less than 4 matches, we will give up. This rule will quickly eliminate many false matches, but it will also eliminate some of the true matches.

The early jump out rule used by the landmark detection module has the following form: the match function will be terminated if

$$nS_n < \begin{cases} n & \text{if } n < \zeta N \\ NS_{\min} - (N - n) & \text{otherwise} \end{cases} . \tag{8.5}$$

Qualitatively, the rule simply states that the first ζ percent of cells that are evaluated must all match; after that, Equation 8.4 is applied to eliminate impossible matches. The rule is parameterised by ζ, which is the called the *strength* of the rule. Higher values of ζ produce a rule that is more likely to reject false matches, but is also more likely to reject true matches. Note that, for this rule to work as intended, cells must be evaluated in a random order. Otherwise, this rule is equivalent to specifying a fixed subset of the cells in the template, all of which must match exactly. This is not the desired result.

Unfortunately, it is not possible to determine, a priori, how often or how quickly an early jump out rule will eliminate false matches. Neither is it possible to determine, in general, how often an early jump out rule will eliminate *true* matches.[3] The effectiveness of the rule described in Equation 8.5 must therefore be established experimentally.

8.3.5 Measuring Performance

There are two key measures of performance for the landmark detection module:

- The landmark detection rate, which measures the average number of landmarks detected per second.

[3] It is possible, however, to place an *upper bound* on the probability that a given early jump out rule will reject a true match if one makes some assumptions about the matching process. See [56] for example.

- The probability of false matches, which measures the proportion of the detected landmarks that represent incorrect matches.

Naturally, we would like to maximise the former and minimise the latter.

Ideally, we would also like the measure the fraction of *actual* landmarks detected: that is, of all the 'true' landmarks in the environment, how many does the system detect? Unfortunately, there are a number of difficulties associated with this measurement. Firstly, the number of detected landmarks depends upon the speed with which the robot is moving – the fraction of landmarks detected will be higher for slow moving robots than for fast moving robots. Secondly, the classification of 'true' landmarks is somewhat subjective – for example, in the case of extended landmarks, such as walls, how does one determine the number of 'true' landmarks? It is for these reasons that the fraction of detected landmarks has not been measured in the experiments described in this chapter.

The landmark detection rate depends on a number of factors: the size and resolution of the local occupancy map, the average time taken to evaluate the match function, and the probability of rejecting a *true* match. We can write down the following relationship:

$$\text{Rate} \propto \frac{1-p}{n_{\text{map}} \times n_{\text{ave}}} \tag{8.6}$$

where p is the probability of rejecting a true match, n_{map} is the number of cells in the local map, and n_{ave} is the average number of cells evaluated by the match function. Of these three terms, only the number of cells in the local map can be determined a priori. The probability of rejecting a true match, and the average number of cells evaluated by the match function, will both depend upon the strength of the early jump out rule. For example, with no early jump out, p and n_{ave} are both constant: $p = 0$ and $n_{\text{ave}} = N$. With early jump out, the value of p and n_{ave} will depend upon the nature of the environment: if the environment contains many objects that are very similar to, but not quite the same as, the target landmarks, one would expect the average number of cells evaluated to be quite high. Alternatively, if the environment consists mainly of objects that are very dissimilar to the target landmarks, one would expect that, on average, fewer cells would be evaluated.

Equation 8.6 highlights the importance of the early jump out rule and its somewhat paradoxical consequences. If, by selecting an early jump out rule that *increases* the probability of rejecting true matches, we cause a larger *decrease* in the average number of cells evaluated, the rate which landmarks are detected may actually *increase*. Consider, for example, the difference between an early jump out rule for which ($p = 0, n_{\text{ave}} = 50$), and another rule for which ($p = 0.5, n_{\text{ave}} = 5$). Clearly, the latter rule will lead to a much high landmark detection rate, even though there is a 50% chance of failing to detect a true match in any given comparison. As I will show in the next section, such results are not atypical.

The other important measure of performance is the probability of false matches. This probability is dictated almost entirely by the match threshold value. A high match threshold leads to a low probability of false matches, but *reduces* the rate at which landmarks detected (by increasing the average number of cells n_{ave} that must be evaluated); a low threshold leads to a high probability of false matches, but *increases* the rate at which landmarks are detected (by decreasing the average number of cells n_{ave} that must be evaluated). Thus, one can choose to have either small amounts of reliable data, or large amounts of unreliable data. Which of these situations is preferable depends on how the information is used. Generally speaking, the global localisation module – which uses landmarks to determine the robot's location with respect to a global map – works best when the probability of false matches is low. Therefore, the match threshold is usually set quite high (typically on the order of 0.90 to 0.95), regardless of the effect this has on the landmark detection rate.

8.4 Experiments

As noted in Section 8.3, there are two key performance measures for the landmark detection module: the landmark detection rate and the probability of false matches. Unfortunately, in practice, it is difficult to accurately quantify either of these measures – their values depend strongly on the particular landmark templates used, the CPU power available to the landmark detection module, the nature of the environment the robot inhabits, and the quality of the local occupancy map. The two experiments presented in this section therefore have a more limited aim:

- The first experiment looks at the *relative* landmark detection rate produced by different early jump out rules. This experiment is intended to establish the utility of the early jump out technique.

- The second experiment presents the performance measures obtained for a typical experimental run through a series of office corridors. This experiment is intended to demonstrate the *characteristic* performance of the landmark detection module.

8.4.1 The Landmark Detection Rate

The relationship between the landmark detection rate and the strength of the early jump out rule is depicted in Figure 8.3. The graph shows the *improvement* in

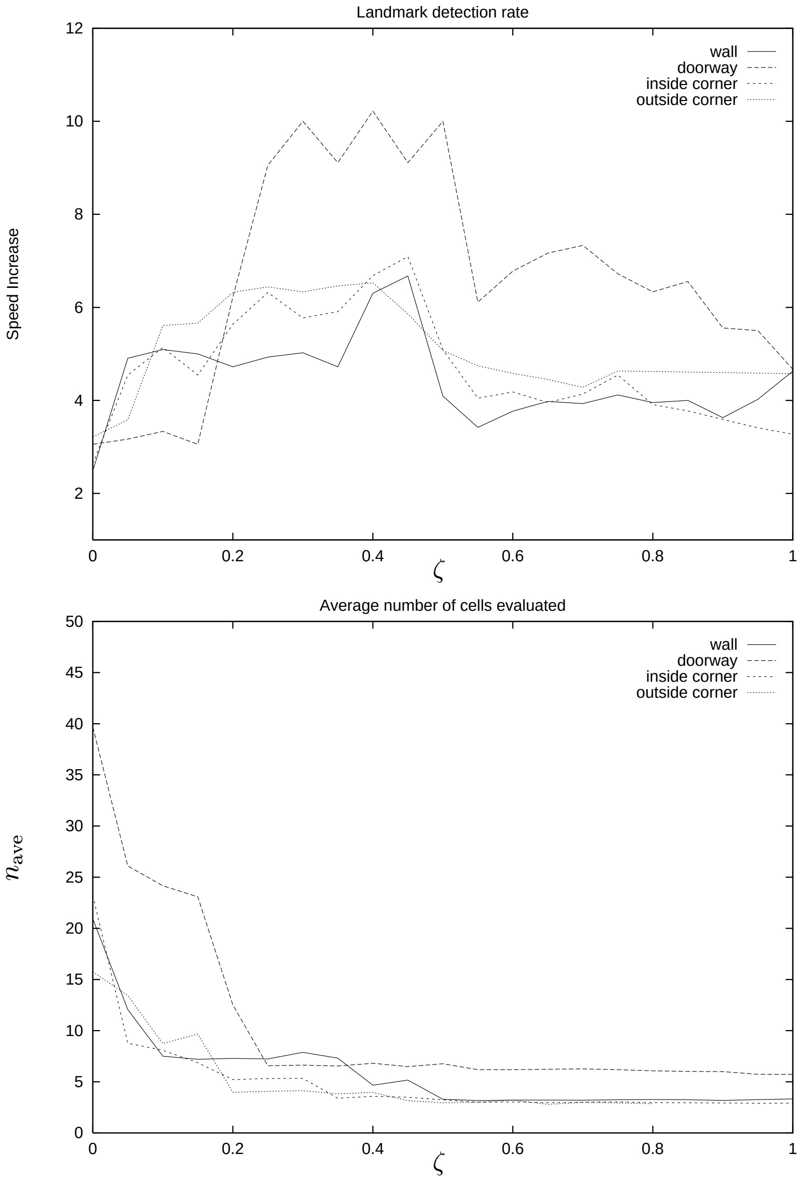


Figure 8.3: (Top) Graph showing the relationship between the landmark detection rate and the early jump out parameter ζ. The vertical axis indicates the improvement in speed over a match function without early jump out. (Bottom) Graph showing the average number of cells evaluated as a function of the early jump out parameter ζ.

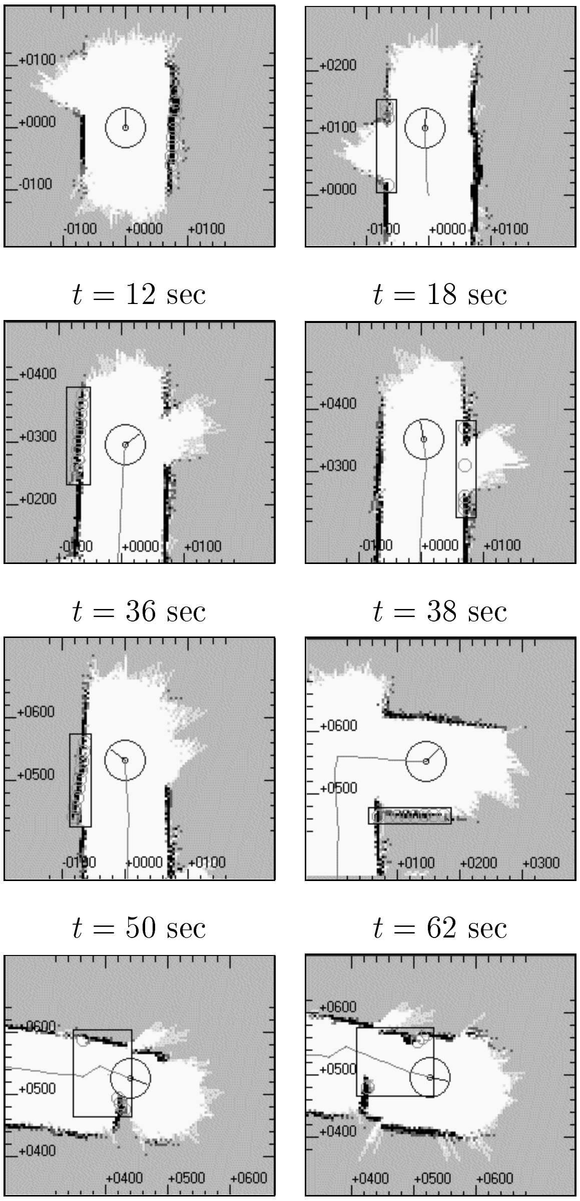
the landmark detection rate as a function of ζ; that is, it compares the landmark detection rate obtained using the early jump out rule against that obtained *without* an early jump out rule. These results were obtained using the landmark templates and occupancy maps shown in Figure 8.1; in all cases, the match threshold $S_{\min}$ was set to 0.90. Each data point in this graph represents the average result obtained after 1000 trials.

Consider the result for the inside corner template. At $\zeta = 0$, which corresponds to the weakest possible early jump out rule, there is a two-fold improvement in the landmark detection rate. This increases to a *five-fold* improvement for $\zeta = 0.55$, which corresponds to an early jump out rule in which the first 55% of the cells evaluated by the match function must all match. The results for other landmarks are similar or better.

The reason for this improvement becomes clear if one inspects the second graph shown in Figure 8.3. This graph shows the average number of cells evaluated by the match function. The results here are quite remarkable: even with relatively weak early jump out rules, the vast majority of potential matches are eliminated after evaluating only a few cells.

8.4.2 Results for a Typical Run

Figure 8.4 shows a series of snapshots produced by J Edgar during a typical experimental run along a corridor. Each map shows the location of a detected landmark. The statistics for the overall run are given in Table 8.1. The robot detected 29 landmarks in total, with two of these being false matches (the pair of doorways at $t = 80$ sec and $t = 92$ sec). The robot also failed to detect either of the outside corner landmarks at $t = 62$ sec.



$t = 12$ sec | $t = 18$ sec

$t = 36$ sec | $t = 38$ sec

$t = 50$ sec | $t = 62$ sec

$t = 80$ sec | $t = 92$ sec

Figure 8.4: Snapshots taken from a typical experimental run. Each snapshot shows the local occupancy map, with the location of a detected landmark indicated by the open rectangle.

Total landmarks detected	29
False landmarks detected	2
Elapsed time (sec)	104
Landmark detection rate (landmarks/sec)	0.28
Probability of false matches (percent)	7

Table 8.1: Results for an experimental run with J Edgar. Note that these results where obtained with an early version of the local navigation module - hence the lack of smooth cornering.

Note that the failure to detect landmarks is not really a problem for the overall navigation system. Since the global localisation module uses only those landmarks that are *detected* for determining the robot's location, the *failure* to detect a landmark has relatively little effect on the localisation process (of course, if very few landmarks are detected, the robot will be unable to effectively localise itself). During the course of this run, more than enough landmarks were detected to ensure accurate localisation.

On the other hand, the two false matches at $t = 80$ sec and $t = 92$ sec *do* present something of a problem, since there will be no corresponding landmarks in the global model. It is for this reason (among others) that the localisation module employs a *probabilistic* localisation technique – it makes the localisation process comparatively robust to the presence of false matches. Note that, in subsequent experiments, it was found that the false matches shown in Figure 8.4 could be eliminated by making some relatively minor changes to the doorway landmark template.

8.5 Conclusion and Further Work

The approach described in this chapter is both fast and robust, principally as a result of the early jump out technique used to increase the speed of the matching process. Furthermore, while the landmark detection module was not designed to operate in dynamic environments, it has shown itself to be quite robust to the presence of moving obstacles, such as people. Generally speaking, the presence of such obstacles simply decreases the landmark detection rate – it is hard to recognise a doorway, for example, when there is someone standing in it. On the other hand, such obstacles can occasionally cause the robot to mis-identify a landmark, with potentially serious consequences for the localisation process. On one occasion, for example, the landmark detection module decided that a group of people standing in a line was a wall, resulting in a failure of the localisation process. Fortunately, such events are quite rare.

There are two natural extensions to work described in this chapter. Firstly, one could extend the landmark detection process to detect structures that *might be* landmarks, as opposed to structures that are *definitely* landmarks. This could be very useful if the robot were equipped with more sophisticated planning algorithms, ones that actually plan the robot's *perceptual* activities. For example, if the robot's location is highly uncertain, and something that might be a landmark is detected, it is probably worth stopping and inspecting the potential landmark. If, on the other hand, the robot's location is highly certain, the potential landmark can be safely ignored.

Secondly, the landmark detection process, as described, is entirely bottom-up. One could also add a top-down component, in which the localisation module feeds the landmark detection module information about where it *expects* to see landmarks. Almost certainly, this would have the effect of increasing the landmark detection rate. Unfortunately, it would also defeat the purpose of the probabilistic localisation approach described in the next chapter, which relies on the (near) statistical independence the measurements made by the landmark detection module for its effectiveness.

Chapter 9

Global Navigation

> "The fact is." said Rabbit, "we've missed out way somehow."
>
> They were having a rest in a small sand-pit on the top of the Forest. Pooh was getting rather tired of that sand-pit, and suspected it of following them about, because whichever direction they started in, they always ended up at it, and each time, as it came through the mist at them, Rabbit said triumphantly, "Now I know where we are!" and Pooh said sadly, "So do I," and Piglet said nothing. He had tried to think of something to say, but the only thing he could think of was, "Help, help!" and it seemed silly to say that, when he had Pooh and Rabbit with him.

from *The House at Pooh Corner*, by A. A. Milne.

Abstract

This chapter describes the three key modules in the global layer – the modelling, localisation and planning modules. Collectively, these modules are responsible for solving global *navigation tasks.*

9.1 Introduction

This chapter addresses the problem of global navigation. In its most conventional form, this problem can be stated as follows – how does a robot, whose initial location is known, go about reaching a distant goal? This chapter, however, addresses a somewhat more difficult problem – how does a robot, whose initial location is either highly uncertain or *completely unknown*, go about reaching a distant goal?

Modelling, Localisation and Planning

Traditional approaches to landmark-based navigation generally divide the overall problem into three processes – modelling, localisation and planning. These processes are defined as follows:

- Modelling is the process whereby the robot builds a model of its environment. This model will contain at least two kinds of information: it will describe the distribution of landmarks within the environment, and it will describe the connectivity of the environment (i.e. how the robot can move from one location to another).

- Localisation is the process whereby the robot determines its location with respect to the model. Typically, this is done by matching the landmarks observed by the robot against the landmarks recorded in the model.

- Planning is the process whereby the robot attempts to find a series of actions that will take it from its current location to the goal location.

Unfortunately, these approaches are not able to solve the more difficult navigation problem posed above, since they assume that there is relatively little uncertainty in the robot's location. It is for this reason that the navigation system described in this chapter makes use of a *probabilistic* approach. In this approach, the processes of modelling, localisation and planning are defined as follows:

- Modelling is the process whereby a human operator provides the robot with an a priori model of the environment (this thesis does not address the problem of model learning).

- Localisation is the process whereby the robot infers the *set of locations* the robot might have. In practice, this means constructing a *probability distribution* over the space of all possible locations.

- Planning is the process whereby the robot tries to find a series of actions that will not only take the robot to the goal, but will allow the robot to *know* that it has reached the goal. That is, by the time the robot reaches the goal, there should be very little uncertainty in its location.

As I will show, there are a number of difficulties with this approach. Firstly, one must find an efficient mechanism for encoding and updating the *continuous* probability distribution generated by the localisation process. I will discuss two mechanisms for doing this in Section 9.5. Secondly, the generalised planning problem defined above is extremely difficult to solve. Therefore, the probabilistic navigation approach makes use of the concept of a *navigation strategy*. A navigation strategy represents a partial solution to the planning problem – one that will enable to robot to reach the goal most of the time, and in most environments. In Sections 9.6 and 9.7, I will consider a number of such strategies, and compare their performance in an experimental context.

Assumptions

Before proceeding with the remainder of this chapter, it is important to make explicit a number of assumptions about the robot and the environment it inhabits. Specifically:

- The robot is able to detect landmarks and measure their properties.
- The robot is equipped with odometry (or some similar sensor) that allows it to make estimates of its motion.
- The robot is not equipped with an absolute position sensor, such as GPS.
- The robot is not equipped with an absolute orientation sensor, such as a compass.

Note that there is a further assumption here – that the landmarks detected by the robot are intrinsically *ambiguous*. For example, the robot might be able to detect a doorway, but it cannot read the room number. As a consequence, the robot may have to undertake a complex series of actions to determine its location.

There is one further assumption:

- The environment is (mostly) static.

This assumption is made more for the sake of simplifying the following discussion than out of strict necessity. The navigation system described in this thesis is quite capable of handling some forms of variation in the environment. For example, it is not affected by moving obstacles, such as people, unless they actively set out to block the robot's path. Nevertheless, the following discussion will focus primarily on static environments.

Chapter Structure

The remainder of this chapter is structured as follows. Section 9.2 discusses some related work, and gives some background to the techniques described in this chapter. Section 9.3 presents some mathematical preliminaries that are necessary for a proper understanding of the sections that follow. Sections 9.4, 9.5 and 9.6 describe, in detail, the three modules that are collectively responsible for global navigation: the modelling, localisation and planning modules. Finally, Section 9.7 presents a range of experimental results. In particular, it compares the effectiveness of a number of different navigation strategies in solving the navigation problem posed above.

Note that a number of early versions of the work described in this chapter can be found in [120], [122] and [121].

9.2 Related Work

Modelling

The model used to support global navigation can come from one of two sources: it can be provided by a human operator, or it can be learnt by the robot itself. The problem of model learning is, unfortunately, beyond the scope of this thesis. Consequently, it is assumed that a human operator provides the robot with a model of the environment that is both complete and (reasonably) accurate. However, for the sake of completeness, I will review some of the recent approaches to model learning.

Model learning is a fundamentally difficult problem: in order to know its location, the robot needs a map of the environment, but in order to generate a map, the robot needs to know it location. Hence, this problem is sometimes referred to as SLAM, for simultaneous localisation and mapping. It is a problem that has attracted widespread attention [11, 67, 68, 139, 243, 246, 244, 259, 260, 186].

One of the most interesting recent efforts is due to Thrun [243, 246], who presents two alternative approaches to model learning. The first approach [246] involves a semi-autonomous procedure, in which a human operator selects a small set of significant locations in the environment. Whenever the robot arrives at one of these locations, the operator 'presses a button' to inform the robot. Using this relatively sparse information, the robot is then able to construct a detailed map of its environment.

The second approach is entirely autonomous [243], with the robot using an *exploration algorithm* to systematically explore its environment. This approach does, however, make some fairly strong assumptions about the environment the robot inhabits. Specifically, it assumes that the environment is composed entirely of extended linear surfaces (walls) having only four possible orientations. It is not clear the extent to which this approach can be applied to environments in which this assumption is violated.

Despite the success of these techniques (the latter approach has been used to build maps measuring up to 60m by 20m), it is clear that much work remains to be done in this area.

Localisation

Localisation is a key problem for mobile robot navigation, for which many solutions have been proposed. Some of these solutions focus on *static* localisation, i.e. determining the robot's location entirely on the basis of the *currently* observed features of the environment [234, 235, 236, 226]. These approaches have much in common with object pose determination techniques used in computer vision [29]. Others focus on *dynamic* localisation, i.e. on integrating the information obtained by the robots sensors with odometric information to generate a continuously updated estimate of the robots location [60, 72, 157, 158, 259, 260]. The majority of these approaches employ some form of statistical procedure: maximum likelihood estimates and Kalman filters [32] are commonly used. These approaches are, however, quite limited, since they generate only a *single* estimate of the robot's location, and hence require a good initial estimate of this location. If this estimate becomes corrupted, for whatever reason, these systems will fail.

Of more interest are the *probabilistic* localisation techniques that have recently been developed [224, 140, 243, 246, 244, 44, 130], in which the robot maintains a *probability distribution* over the space of all possible robot locations.[1] This approach offers a much more robust solution, since it does not require good initial estimate of the robot's location.

An early version of this approach, due to Simmons and Koenig [224, 140], makes use of topological maps of the environment. That is, the environment is represented as a graph in which nodes correspond to 'locations' and arcs correspond to 'actions'. The representation is quite coarse: a node might correspond to an area several meters across. Unfortunately, as I will show in Section 9.5, there are number of severe limitations associated with such coarse-grained representations; in particular, they are not very good at capturing small changes in the robot's motion.

More recent approaches, due to Thrun [243, 246, 244] and Burgard [44], have employed metric, grid-based representations of the environment. These representations tend to be quite fine: each cell in the grid typically represents an area less than 10cm across. While these grid-based approaches are much more computationally expensive than their graph-based counterparts, they have also produced very impressive results. In Section 9.5, I will consider the grid-based representation in more detail. In addition, I will introduce an alternative representation, called the *population-based* representation, which has a number of advantages over the grid-based representation.

[1] These techniques are often referred to as *Markov* localisation, since they can be derived from the theory of Markov Random Fields [86]. The same results can, however, be derived from first principles using simple Bayesian analysis. Consequently, in this thesis, the term *probabilistic* localisation is used.

Planning

There is a vast body of literature devoted to the problem of path planning for mobile robots. Traditional methods include A* search through visibility graphs [200] or configuration space [161]. Techniques based on distance transforms [127, 265] and harmonic functions [264] are also relatively popular.

All of these techniques assume that the location of the robot is known, at least approximately. Unfortunately, with a probabilistic localisation approach, it is not always possible to make this determination. Sometimes, the best one can do is indicate that the robot is at one of a *number* of possible locations. Consequently, in this context, the relevance of these techniques is debatable.

Very few authors have paid serious attention to the problem of planning under these circumstances. The two notable exceptions are Kaelbling [130] and Koenig [140]. Both authors adapt results from Markov Decision Process theory (MDP) to define *policies* for the robot to follow. These policies share much in common with the navigation strategies described in this chapter; these policies are described in detail in Section 9.6.

There are a number of other approaches to planning that are worthy of mention here, even though they do not fall into the category of probabilistic approaches. To some extent, however, these approaches anticipate the ideas presented in this chapter or are aimed at solving similar problems.

Dudek *et al* [69], describe an algorithm for explicitly re-localising a robot. This is a theoretical analysis that describes how a robot at some unknown location in a polygonal environment can determine its location with the least possible effort. That is, it describes an algorithm for obtaining the shortest path the robot can take to unambiguously determine its location. This work is attractive because of the theoretical guarantees it places on the robot's performance; unfortunately, it is unclear how the technique can be applied to real robots, where the uncertainties associated with both sensing and action must be considered.

Nourbakhsh *et al* [182] describe an mobile robot architecture that makes heavy use of *assumptions* to simplify the planning process. In this approach, the robot will make plausible assumptions about the uncertainty in initial conditions, actions and perceptions. Importantly, these uncertainties are not simply 'assumed away' to leave a completely deterministic system. Instead, the robot monitors each plan as it is being executed, looking for evidence that one or more of the assumptions under which the plan was formulated is false. If an assumption is falsified, the robot must re-plan. This approach has been successfully implemented on a number of robots, including 'Dervish', the winner of the AAAI'94 contest.

Another approach to planning for navigation problems is described by Koenig and Simmons [138]. The authors take a class of real-time search algorithms, which are normally applied to AI problems in deterministic domains, and extend these algorithms to handle non-deterministic domains, such as robot navigation. An important feature of real-time search algorithms is that they do not attempt to consider every possible contingency, and consequently do not necessarily produce an optimal result. Instead, they consider as many contingencies as possible given the limited time and computational resources available, in order to arrive at a sub-optimal, but nevertheless workable, solution. While this approach is very interesting, it does not appear have been tested against the kind of navigation problems described in this chapter.

Kuipers *et al* [151] describe a method for exploration and navigation in the context of topological maps. During the map building process, the possibility exists that the robot will arrive at a location whose sensor readings are identical to a previously visited location, leading to a possible ambiguity in the map. To resolve this ambiguity, the robot adopts a fairly simple procedure. Under the assumption that it is at the previously visited location, it plans a return trip via a number of adjacent locations. If the plan can be executed successfully, the robot assumes that it is at a previously visited location. If the plan cannot be executed successfully, the robot assumes it is at a new location. There are, of course, pathological environments in which this procedure will fail. However, for most realistic environments, the procedure is quite robust. Interestingly, the approach described in this chapter produces very similar robot behaviour; this behaviour is, however, an *emergent* property of the formalism, rather than something that has been explicitly encoded.

A different approach is taken by Takeda *et al* [233], who describe a planning technique that takes account of the *sensory uncertainty* associated with different parts of the environment. That is, they describe an algorithm for finding paths that minimise localisation errors. This particular idea would provide a very useful extension to the work described here; it would allow the robot to automatically avoid situations that are likely to result in the robot becoming lost.

9.3 Some Mathematical Preliminaries

In the following sections, we will take a detailed look at the three modules that make up the global layer. First, however, we must take a couple of minor detours, and consider the problem of *probabilistic localisation* and of *coordinate transformations*.

9.3.1 Probabilistic Localisation

The global localisation module makes use of a *probabilistic* localisation approach. In this approach, the robot does not attempt to estimate its location directly. Instead, it maintains a *probability distribution* $p(r)$ over the space of all possible locations. Note that, throughout this chapter, the term *location* will be used to describe the robot's location *and* orientation; hence, $r = (r_x, r_y, r_\theta)$, where (r_x, r_y) denotes the robot's location (in a two dimensional space), and r_θ denotes the robot's orientation. Two different kinds of measurements are used to update the distribution $p(r)$: unary measurements, which yield information about the robot's *current* location; and binary measurements, which yield information about the *change* in the robot's location. The detection of a landmark, for example, would constitute a unary measurement. Similarly, a change in the robot's position – as determined by odometry – would constitute a binary measurement.

In the case of unary measurements, the probability $p(r)$ can be updated using Bayes' Law:

$$p(r \mid u) = \frac{p(u \mid r)}{p(u)} p(r) \tag{9.1}$$

where $p(r \mid u)$ denotes the probability that the robot is at location r, given that it has made the measurement u; and $p(r)$ is the *prior* probability that the robot is at location r. The denominator in this expression can thought of as a normalisation term – i.e. $p(u)$ must be such that

$$\int p(r \mid u) dr = 1. \tag{9.2}$$

The key term in Equation 9.1 is the numerator: $p(u \mid r)$. This term – which gives the probability that the robot will make the measurement u, given that it is at location r – is called the *unary sensor model*, and forms part of the robot's overall environment model. The unary sensor model can be used to encode many different kinds of information. For example, it can be used to:

- Exclude certain locations. If $p(u \mid r) = 0$ for *all* u, there is no way the robot can be at r. This allows us to specify, for example, that the robot cannot occupy the same volume as a wall.

- Describe the visibility and distribution of landmarks within the environment. That is, the model records which landmarks are visible when the robot is at r, and their expected position and orientation with respect to the robot.

- Describe the uncertainty in the *measurements* made by the robot. For example, if the robot has detected a landmark at some distance from the object, the uncertainty associated with this measurement can be encoded in $p(u \mid r)$.

- Describe the uncertainty in the *model*. Typically, the model is prepared by a human operator working from a set of floor-plans. These plans are often quite inaccurate (the builders sometimes take a liberal interpretation of the architect's designs). Such inaccuracies can be expressed as *uncertainties* through $p(u \mid r)$.

The unary sensor model must either be supplied to the robot by a human operator, or it must be learnt by the robot itself. In this thesis, it is assumed that the former is the case.

In the case of binary measurements, the probability $p(r)$ is updated using a slightly more complex expression:

$$p(r \mid b) = \int p(r, r' \mid b) p(r') dr' \tag{9.3}$$

where $p(r, r' \mid b)$ measures the probability that the robot has moved from r' to r, given that it has made the measurement b. This expression computes the probability that the robot is at r by considering all the locations that the robot *might have been at*, and all of the ways in which its location *might have changed*. The term $p(r, r' \mid b)$ is calculated by employing Bayes' Law once again:

$$p(r, r' \mid b) = \frac{p(b \mid r, r')}{p(b)} p(r, r') \tag{9.4}$$

Typically, it is assumed that the prior probability $p(r, r')$ is constant for all (r, r'), in which case Equation 9.3 simplifies to:

$$p(r \mid b) = \int \frac{p(b \mid r, r')}{p(b)} p(r') dr' \tag{9.5}$$

where $p(b)$ is a normalisation constant.

The term $p(b \mid r, r')$ – which gives the probability that the robot will make the measurement b, given that it has moved from r' to r – is called the *binary sensor model*. Like the unary sensor model, this term forms part of the robot's overall environment model. The binary sensor model can be used to:

- Exclude certain changes in the robot's location. If $p(b \mid r, r') = 0$ for all b, there is no way the robot can move from r to r'. This allows us to specify, for example, that the robot cannot move through a wall.

- Describe the uncertainty associated with the *measurements* made by the robot. Odometric drift, for example, can be encoded in this way. Moreover, we can specify that odometry is more accurate in some parts of the environment than others (odometry is generally more accurate on smooth surfaces than on rough ones).

- Describe the uncertainty in the model itself. For example, the connectivity of the environment may not be completely understood.

As with the unary sensor model, it is assumed that this model is provided by a human operator.

An important assumption in the probabilistic localisation approach is that measurements are *statistically independent*. Formally, this means that the probability of obtaining the pair of measurements (u_1, u_2) is equal to the product of their individual probabilities. That is:

$$p(u_1, u_2 \mid r) = p(u_1 \mid r) \cdot p(u_2 \mid r). \tag{9.6}$$

From this, it follows that the probability that the robot is at r, given a *pair* of measurements (u_1, u_2), is given by:

$$\begin{aligned} p(r \mid u_1, u_2) &= \frac{p(u_1, u_2 \mid r)}{p(u_1, u_2)} p(r) \\ &= \frac{p(u_1 \mid r)}{p(u_1)} \frac{p(u_2 \mid r)}{p(u_2)} p(r) \end{aligned} \tag{9.7}$$

The unary update formula (Equation 9.1) can then be recast as an *incremental* update rule – given some measurement u, the distribution $p(r)$ is updated using:

$$p(r) \xleftarrow{u} \frac{p(u \mid r)}{p(u)} p(r). \tag{9.8}$$

Naturally, this argument holds for binary measurements also; given some binary measurement b, the distribution $p(r)$ is updated using:

$$p(r) \xleftarrow{b} \int \frac{p(b \mid r, r')}{p(b)} p(r') dr'. \tag{9.9}$$

Together, Equations 9.8 and 9.9 form the core of the probabilistic localisation approach – they allow the distribution $p(r)$ to be updated quickly and efficiently whenever the robot makes a new measurement.

Note that there is one caveat on the above results – they apply only if the statistical independence assumption holds true. In practice, this is not entirely unproblematic. For example, if a sensor were to generate multiple *identical* measurements of the same landmark, this would tend to skew the distribution. In effect, too much weight would be given to some measurements and not enough to others. Consequently, care must be taken to ensure that the measurements generated by the robot's sensors conform to the statistical independence assumption.

As an illustration of the probabilistic localisation approach, consider the series of images shown in Figure 9.1. These images show the evolution of the distribution $p(r)$ for a robot moving through a simulated environment. Initially, the robot's location is unknown, as indicated by the uniform distribution in Figure 9.1(a). Soon, however, the detection of landmarks reduces the number of possible robot locations – the distribution in Figure 9.1(b) has formed a number of distinct peaks. Eventually, the robot has detected enough landmarks to unambiguously determine its location – the distribution in Figure 9.1(c) shows a single sharp peak.

In this discussion of the probabilistic localisation approach, the problem has been framed entirely in terms of *continuous distributions*, i.e. in terms of distributions defined over a three-dimensional real-valued space. In some previous work on probabilistic localisation, the problem has been treated in a discrete fashion – see, for example, the work of Simmons [224] and Koenig [140]. Unfortunately, these discrete approaches have a serious problem – they are not very good at capturing the effects of *changes* in the robot's location, particularly when odometry is used to measure these changes. Consequently, the following claim is made: at a theoretical level, probabilistic localisation should be treated as a *continuous* problem, despite the fact that, in practice, it must be implemented using a discrete representation.

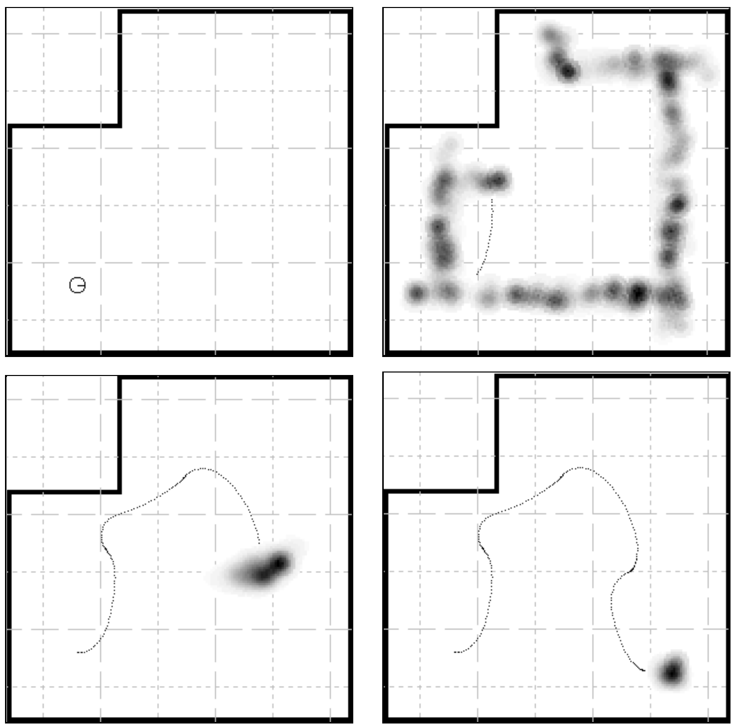


Figure 9.1: Evolution of the location distribution in an environment containing wall and corner landmarks. Dark regions indicated locations at which the robot is more likely to be found. The robot's initial location is unknown.

To put this another way: the discrete representations used to implement the probabilistic localisation approach should be seen as *approximations* to the continuous distributions of the formalism. Sections 9.4 and 9.5 describe a number of discrete representations, both for the model and the distribution $p(r)$.

9.3.2 Coordinate Transformations

The modules in the global layer receive information from, and send commands to, modules in the local layer. Unfortunately, these two layers use completely different coordinate systems. The global layer uses the *global coordinate system*, which is fixed with respect to the environment. All modelling, localisation and planning is carried out with respect to this coordinate system. The local layer, on the other hand, uses the *odometric coordinate system*, whose origin with respect to the environment is entirely arbitrary. If one discount the effects of odometric drift, the origin of the odometric coordinate system corresponds to the location of the robot when it was first switched on. The landmarks located by the landmark detection module are described in terms of their position and orientation with respect to this coordinate system, as are the goals that are sent to the local navigation module. For shorthand, we will refer to the global and odometric coordinate systems as the GCS and the OCS, respectively.

The first challenge for the modules in the global layer is to find a transformation that maps points in the OCS into points in the GCS, and vice-versa. We write this transform as Γ. Given a point $\hat{x} = (\hat{x}_x, \hat{x}_y, \hat{x}_\theta)$ in the OCS, the corresponding point $x = (x_x, x_y, x_\theta)$ in the GCS is given by:

$$x = \Gamma \hat{x} \tag{9.10}$$

Clearly, there is no way of knowing the transform Γ a priori (since this would be equivalent to solving the localisation problem). However, let us assume for a moment that we know the robot's location $r = (r_x, r_y, r_\theta)$ in the GCS, and its corresponding location $\hat{r} = (\hat{r}_x, \hat{r}_y, \hat{r}_\theta)$ in the OCS. In this case, Γ must be such that:

$$\begin{bmatrix} x_x \\ x_y \\ x_\theta \end{bmatrix} = \begin{bmatrix} r_x \\ r_y \\ r_\theta \end{bmatrix} + \begin{bmatrix} \cos r_\theta & -\sin r_\theta & 0 \\ \sin r_\theta & \cos r_\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \hat{r}_\theta & \sin \hat{r}_\theta & 0 \\ -\sin \hat{r}_\theta & \cos \hat{r}_\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \left(\begin{bmatrix} \hat{x}_x \\ \hat{x}_y \\ \hat{x}_\theta \end{bmatrix} - \begin{bmatrix} \hat{r}_x \\ \hat{r}_y \\ \hat{r}_\theta \end{bmatrix} \right) \tag{9.11}$$

This transform is easier to understand if one breaks it into two parts: the first part maps points from the OCS into a *robot centred* coordinate system; the second part then maps these points into the GCS. For brevity, we will write this transform as $\Gamma_{r\hat{r}}$; this transform will be used over and over again in the following sections. Note that $\Gamma_{r\hat{r}}$ is a time-dependent transform – not only will $\hat{r}$ and r will change over time, odometric drift will ensure that the relationship between them also changes over time. Consequently, when using this transform, one must take care to use consistent values for $\hat{r}$ and r.

The transform $\Gamma_{r\hat{r}}$ has two important properties. Firstly, as one would expect, it maps the robot's location $\hat{r}$ in the OCS onto the corresponding robot location r in the GCS:

$$r = \Gamma_{r\hat{r}}\hat{r} \tag{9.12}$$

Secondly, the inverse transform $\Gamma_{r\hat{r}}^{-1}$ is given by:

$$\Gamma_{r\hat{r}}^{-1} = \Gamma_{\hat{r}r} \tag{9.13}$$

That is, one can form the inverse transformation by simply swapping the subscripts. Consequently, in addition to mapping a location in the OCS to a location in the

GCS, one can do the reverse:

$$\hat{x} = \Gamma_{\hat{r}r} x \tag{9.14}$$

This is particularly useful for transforming actions. In the global layer, actions are described with respect to the GCS; in the local layer they are described with respect to the OCS. Actions are discussed in detail in Section 9.4.

9.4 The Modelling Module

The modelling module is responsible for maintaining a global model of the robot's environment. In practice, since the model is provided by a human operator, the modelling module is simply a complex data structure. From the point of view of other modules, the modelling module is a 'black-box': given an appropriate set of inputs, it will compute a value for any of the terms in the model.

The model is used for both localisation and planning. The localisation process relies on two kinds of information: the type and location of detected landmarks (as determined by the landmark detection module), and change in the robot's location (as determined using odometry). The planning process, on the other hand, only needs to know the probable outcome of any actions the robot may perform. Consequently, the modelling module maintains a model with three terms:

$$\text{Model} = [p(f, x \mid r), p(\Delta r \mid r, r'), p(r' \mid r, a)]. \tag{9.15}$$

The first term (which is also known as the *unary sensor model*) describes the probability of observing a landmark of type f at location x, given that the robot is at location r. This term is used to encode a number of different properties of the environment:

- The visibility and distribution of landmarks, i.e. which landmarks are visible when the robot is at r, and the expected location of those landmarks.

- The uncertainty associated with the landmark detection process.

The second term in the model (the *binary sensor model*) describes the probability that the robot will measure a change in location of Δr (as determined using odometry), given that it has actually moved from location r' to location r. This term encodes two more properties of the environment:

- The connectivity of the environment, i.e. whether of not the robot can reach r' from r.

- The uncertainty associated with odometric measurements.

The final term in the model (the *action model*) describes the probability that the robot will arrive at location r', given that it starts at location r and performs action a. This term encodes:

- The outcome of any actions the robot may perform, i.e. the location the robot expects to reach if it is currently at r and performs action a.

- The uncertainty associated with actions. Since the actions selected by the global layer are passed back down to the local layer for execution, the robot will not always end up at the location that was anticipated.

Note that, in practice, the robot is limited to carrying out one of nine possible actions: it can either come to a complete stop, or it can move in any of the eight cardinal compass directions – north, north-east, east, south-east, and so on.

There is, unfortunately, one complication involved in using this model – all of the terms in the model are defined with respect to the *global* coordinate system (the GCS), but all of the measurements made by the local layer are defined with respect to the *odometric* coordinate system (the OCS). Consequently, when working with this model, one must make appropriate use of the coordinate transform operator $\Gamma_{r\hat{r}}$ (Equation 9.11). For example, if the landmark detection model detects a landmark of type f at location $\hat{x}$ *as measured with respect to the OCS*, the probability of this event is given by:

$$p(f, \hat{x}, \hat{r} \mid r) = p(f, \Gamma_{r\hat{r}}\hat{x} \mid r), \tag{9.16}$$

where the right hand side is the first term in the model. Similarly, if the robot's location has changed by an amount $\Delta\hat{r}$, as measured with respect to the OCS, the probability of this event is given by:

$$p(\Delta\hat{r}, \hat{r} \mid r, r') = p(\Gamma_{r\hat{r}}\Delta\hat{r} \mid r, r'), \tag{9.17}$$

where the right hand side corresponds to the second term in the model. Finally, before an action is sent to the local layer for execution, it must be transformed from an action a that is defined with respect to the GCS, to an action $\hat{a}$ that is defined with respect to the OCS. For this, the inverse transform (Equation 9.13) is used:

$$\hat{a} = \Gamma_{\hat{r}r}a. \tag{9.18}$$

9.4.1 Implementation

As noted in Section 9.4, the modelling module can be regarded as a 'black-box': given an appropriate set of inputs, it will compute a value for any of the terms in the model. For all intents and purposes, this module encodes a set of *continuous* distributions.

Internally, however, the story is quite different: the continuous distributions are represented in a discrete fashion. We will consider two such representations – one grid-based, the other graph-based – and discuss their relative merits.

A Grid-based Representation

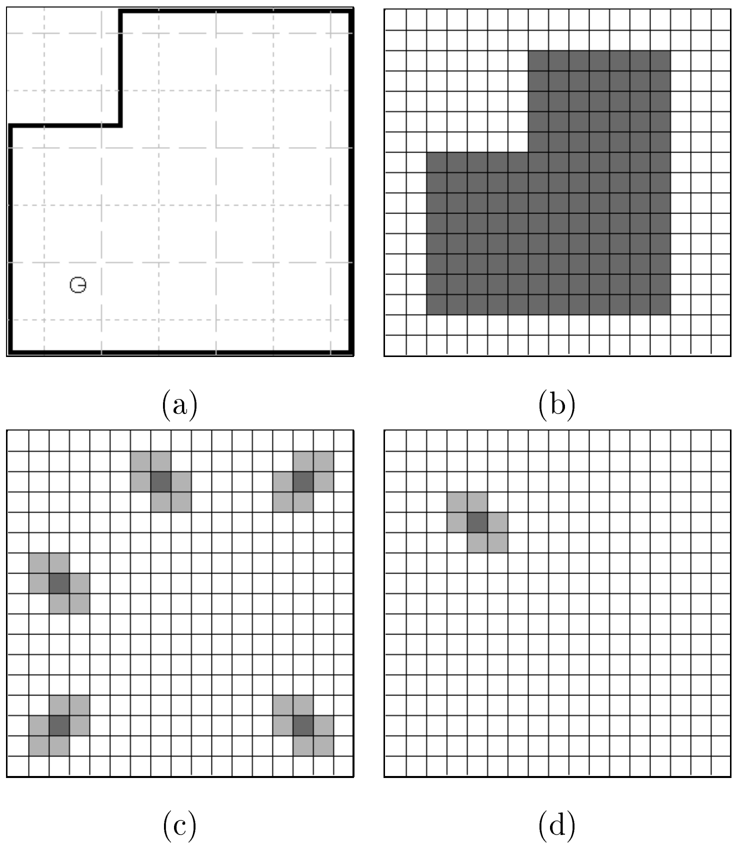
In the grid-based representation, the space of all possible locations is divided into a large number of cells using a regular grid. For the sake of simplicity, a two-dimensional grid is used; the orientation component is ignored. Two maps are defined with respect to this grid:

- The free-space map $\mathrm{Free}(i, j)$. This map stores the probability that cell (i, j) is free space (i.e. can be occupied by the robot).
- The landmark map $\mathrm{Land}(f, i, j)$. This map stores the probability of observing a landmark of type f in cell (i, j).

Consider, for example, the set of images shown in Figure 9.2. Figure 9.2(b) shows the free-space map for a simple simulated environment shown in Figure 9.2(a); Figures 9.2(c) and Figure 9.2(c) show the distribution of inside-corner and outside-corner landmarks, respectively. In the latter maps, the uncertainty associated with the landmark detection process is indicated through the shading of the image – the darker the pixel, the more likely the robot is to observe a landmark in the corresponding grid cell.

In addition to the two grid-based maps, the modelling module also employs a pair of *drift terms*:

- The odometric drift term $\mathrm{Odo}(\delta)$. This term records the uncertainty associated



(a) (b) (c) (d)

Figure 9.2: Grid-based model for a simulated environment. (a) Aerial view of the environment. (b) Free-space map. (c) Landmark map for inside-corners. (d) Landmark map for outside-corners.

with odometric measurements, and is assumed to be Gaussian in form:

$$\mathrm{Odo}(\delta) = \frac{1}{(2\pi)^{3/2}\sigma_{ox}\sigma_{oy}\sigma_{o\theta}} \exp - \left(\frac{\delta_x^2}{2\sigma_{ox}^2} + \frac{\delta_y^2}{2\sigma_{oy}^2} + \frac{\delta_\theta^2}{2\sigma_{o\theta}^2} \right) \tag{9.19}$$

where $\delta = (\delta_x, \delta_y, \delta_\theta)$ is the difference between the actual change in the robot's location, and the change measured using odometry. The distribution is parameterised by the vector $\sigma_o = (\sigma_{ox}, \sigma_{oy}, \sigma_{o\theta})$, whose value must be determined experimentally.

- The actuator drift term $\mathrm{Act}(\delta)$. This term records the uncertainty associated with the outcome of actions; it is also assumed to be Gaussian in form:

$$\mathrm{Act}(\delta) = \frac{1}{(2\pi)^{3/2}\sigma_{ax}\sigma_{ay}\sigma_{a\theta}} \exp - \left(\frac{\delta_x^2}{2\sigma_{ax}^2} + \frac{\delta_y^2}{2\sigma_{ay}^2} + \frac{\delta_\theta^2}{2\sigma_{a\theta}^2} \right) \tag{9.20}$$

  where $\delta = (\delta_x, \delta_y, \delta_\theta)$ is the difference between the location the robot expected to reach, and the location it actually reached. The distribution is parameterised by the vector $\sigma_a = (\sigma_{ax}, \sigma_{ay}, \sigma_{a\theta})$, whose value must be determined experimentally.

Note that these drift terms represent fairly crude approximations to reality. There are, for example, many sources of odometric uncertainty: the finite resolution of the wheel encoders, the unevenness of the surface over which the robot travels, the uncertainty in the robot's physical dimensions, the possibility of wheel slip, and so on. Since many of these phenomena are difficult to characterise analytically, a simple Gaussian approximation is used for all.

The drift terms are used in conjunction with the free-space and landmark maps to compute the three terms in the model (Equation 9.15). The first term in the model (the unary sensor model) is given by:

$$p(f, x \mid r) = \mathrm{Land}(f, x_x, x_y). \tag{9.21}$$

While this expression encodes both the distribution of landmarks and the uncertainty associated with the landmark detection process, it *does not* encode the visibility or orientation of landmarks. That is, it does not take into account whether or not the robot is able to see a given landmark from its current location, and it does take into account the fact that landmarks can have many different orientations. This is a weakness of this particular approach – there is no mechanism by which the visibility of landmarks can be expressed, and the two-dimensional nature of the grid prohibits the encoding of landmark orientations. Note that landmark visibility *could* be encoded if the representation were extended to include a separate map for each landmark, as opposed to each landmark type. However, this particular representation has not yet been implemented.

The second term in the model (the binary sensor model) is given by:

$$p(\Delta r \mid r', r) = \mathrm{Free}(r'_x, r'_y) \cdot \mathrm{Odo}(r + \Delta r - r') \tag{9.22}$$

In this expression, the free-space map has been used to encode the connectivity of the environment. However, the two-dimensional nature of the grid means that the orientation of the robot is ignored. This is entirely satisfactory if the robot is cylindrical in shape, since, in this case, the connectivity of the environment is not affected by the robot's orientation.

The final term in the model (the action model) is given by:

$$p(r' \mid r, a) = \mathrm{Free}(r'_x, r'_y) \cdot \mathrm{Act}(r + a - r'). \tag{9.23}$$

Once again, this expression ignores the orientation of the robot. However, so long as the robot is able to perform in-place turns, this does not represent a problem – the set of *locations* the robot can reach is not affected by its initial orientation.

From the preceding definitions, it should be clear that the grid-based approach fails to encode many significant properties of the environment. This approach does, however, have two key advantages – it is very easy to implement, and it is very easy for a human operator to prepare the model. In practice, this is done by creating a series of images – one for the free space map, and one for each type of landmark – and feeding them into the modelling module. Given a set of floorplans, these images can be prepared quickly and easily using an off-the-shelf image editing program.

A Graph-based Representation

In the graph-based approach, the space of all possible locations is divided into *regions*, each of which is assigned to a node in a graph. Connected regions are denoted by arcs between the corresponding nodes in the graph. Each node in the graph stores three sets of properties:

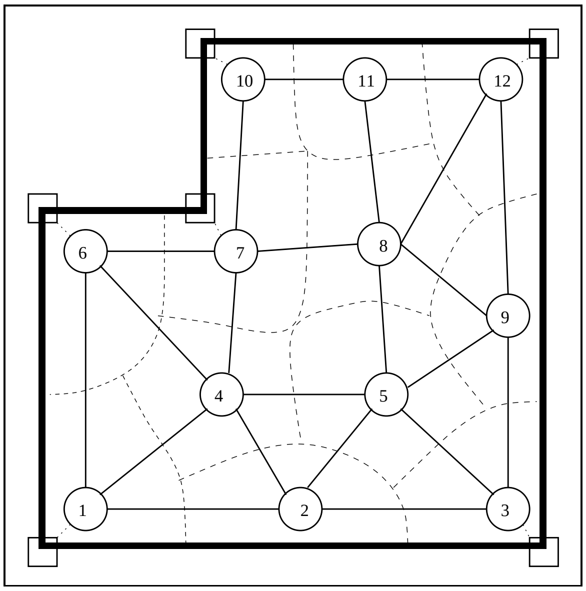
- A *prototype location* r_i, which defines the region associated with node i. A point r is said to belong to node i if

$$i = \arg\min_{i'} |r - r_{i'}|; \tag{9.24}$$

  that is, the point belongs to the node with the nearest prototype. This is a form of Voronoi tessellation [184].

- A list of connected nodes. This list is used to define the *connection function* $\mathrm{Con}(i', i)$, which returns 1 if nodes i and i' are connected, and 0 otherwise.

- A list of landmarks that are visible from this node. For each landmark l that is visible from node i, the list records three properties – the landmark's type f_{il}, its location x_{il}, and the uncertainty associated with its location σ_{il}.




```
LANDMARK Land1 TYPE (InsideCorner)
        AT (0 0 0) SIZE (20 20 15)
LANDMARK Land2 TYPE (InsideCorner)
        AT (300 0 90) SIZE (20 20 15)
LANDMARK Land3 TYPE (InsideCorner)
        AT (300 300 180) SIZE (20 20 15)
LANDMARK Land4 TYPE (InsideCorner)
        AT (100 300 -90) SIZE (20 20 15)
LANDMARK Land5 TYPE (InsideCorner)
        AT (0 200 -90) SIZE (20 20 15)
LANDMARK Land6 TYPE (OutsideCorner)
        AT (100 200 -90) SIZE (20 20 15)

LOCATION Loc1 AT (20, 20)
        CONNECTS (Loc2, Loc4, Loc6)
        LANDMARKS (Land1)

LOCATION Loc2 AT (150, 20)
        CONNECTS (Loc1, Loc3, Loc4, Loc5)
        LANDMARKS ()

LOCATION Loc3 AT (280, 20)
        CONNECTS (Loc2, Loc5, Loc9)
        LANDMARKS (Land2)

LOCATION Loc4 AT (100, 100)
        CONNECTS (Loc1, Loc2, Loc5, Loc6, Loc7)
        LANDMARKS ()

            ⋮
```

(a) (b)

Figure 9.3: (a) Graph-based model for a simulated environment. The dashed lines show the boundaries between regions. Landmarks are indicated by squares, with the dotted lines show the visibility relationships between nodes and landmarks. (b) The text file used to produce the model.

Collectively, these terms are used to define the *landmark distribution function* $\mathrm{Land}(f, x \mid i, l)$, which returns the probability of observing a landmark of type f with location x, given that the robot is at node i and is looking at landmark l. The landmark distribution function is assumed to be Gaussian in form:

$$\begin{aligned} \mathrm{Land}(f, x \mid i, l) \;&=\; \frac{\delta(f, f_l)}{(2\pi)^{3/2}\sigma_{ilx}\sigma_{ily}\sigma_{il\theta}} \\ &\quad \exp - \left[\frac{(x_x - x_{ilx})^2}{2\sigma_{ilx}^2} + \frac{(x_y - x_{ily})^2}{2\sigma_{ily}^2} + \frac{(x_\theta - x_{il\theta})^2}{2\sigma_{il\theta}^2}\right] \end{aligned} \tag{9.25}$$

where the delta function $\delta(f, f_l)$ returns 1 if its arguments are equal and 0 otherwise.

Consider, for example, the model shown in Figure 9.3. This figure shows the graph for a simple environment – the dashed lines indicate the boundaries between regions. In this case, the graph is entirely two dimensional (the orientation component is ignored). Landmarks are indicated by squares, with the dotted lines showing the visibility relationships between nodes and landmarks.

The three terms in the model (Equation 9.15) are computed using the connectivity function $\mathrm{Con}(i, i')$ and the landmark distribution function $\mathrm{Land}(f, x \mid i, l)$. The first term (the unary sensor model) is given by:

$$p(f, x \mid r) = 1 - \prod_l [1 - \mathrm{Land}(f, x \mid i, l)] \tag{9.26}$$

where i is the node computed using Equation 9.24 and the product is taken over all of the landmarks that are visible from node i. This expression takes into account the fact that the landmark observed by the robot may correspond to one of a number of landmarks. That is, Equation 9.26 computes the probability that the robot has observed either landmark 1, *or* landmark 2, *or* landmark 3, and so on. [2] Note that, unlike the grid-based representation, this term encodes both the visibility and orientation of landmarks. It does, however, assume a uniform distribution for landmark visibility – for any given region, each landmark is either entirely visible, or entirely invisible'.

The second term in the model (the binary sensor model) is given by:

$$p(\Delta r \mid r', r) = \mathrm{Con}(i', i) \cdot \mathrm{Odo}(r + \Delta r - r') \tag{9.28}$$

where i and i' are the nodes corresponding to the locations r and r', respectively. The second term in this expression is the odometric drift term described in the previous section – the same drift term is used for both the grid-based and graph-based representations. The connectivity of the environment is encoded through the connectivity function $\mathrm{Con}(i', i)$. This function may or may not encode the orientation of the robot, depending on the set of prototypes used to divide the location space into regions. For example, if all prototypes have the same value for their orientation component, the orientation of the robot is effectively ignored. If, one the other hand, the prototypes have a range of values for their orientation component, the orientation of the robot becomes significant.

The final term in the model (the action model) is given by

$$p(r' \mid r, a) = \mathrm{Con}(i', i) \cdot \mathrm{Act}(r + a - r') \tag{9.29}$$

[2]Given two independent events a and b, the probability of obtaining the event a *or* b is given by:

$$p(a \vee b) = 1 - p(\neg a)p(\neg b) = 1 - (1 - p(a))(1 - p(b)) \tag{9.27}$$

This is a basic result from probability theory.

where the second term is the actuator drift term described in the previous section. As with the binary sensor model (Equation 9.28), this expression may or may not encode the orientation of the robot, depending on the nature of the prototype locations.

Compared to the grid-based representation, the graph-based representation encodes much more information about the robot's environment. In particular, it encodes both the visibility and orientation of landmarks, which can significantly aid the localisation process. On the other hand, constructing a graph-based model is a much more arduous task for a human operator. Presently, the model is created using a text file – such as the one shown in Figure 9.3 – that describes the properties of each and every node in the graph. Preparing these files can be extremely time consuming, particular if the environment is large or complex. It is also very easy to introduce errors.

Choosing a Representation

After some preliminary experiments with both kinds of models, I decided to abandon the use of graph-based models in favour of their grid-based counterparts. Even though this representation is not as rich as the graph-based representation, the ease with which the model can be prepared makes the grid-based representation attractive. On the other hand, if the robot where capable of learning its own model of the environment, a graph-based representation would probably be preferable. An interesting extension to the work described here would involve a direct comparison of the two representations, to assess their relative advantages and disadvantages. All of the experiments described in this chapter, however, were conducted using a grid-based representation.

9.5 The Localisation Module

The localisation module is responsible for maintaining a probability distribution $p(r)$ over the space of all possible locations. Initially, this distribution is configured so that it matches the robot's a priori knowledge about its location. If the robot's location is known *exactly*, $p(r)$ will be a delta function (i.e. infinite at one point and zero everywhere else). If the robot's location is completely *unknown*, $p(r)$ will have a constant value – no location is more likely than any other location. Generally, the initial distribution will lie somewhere between these two extremes.

The localisation module will update the distribution $p(r)$ in response to two different kinds of events: the detection of a new landmark (a unary measurement), and motion of the robot (a binary measurement).

- **Unary Update** Upon detecting a new landmark, the landmark detection module sends three pieces of information to the localisation module: the type f and location $\hat{x}$ of the landmark, and the location $\hat{r}$ of the robot. Both locations are, of course, defined with respect to the OCS. In response, the localisation module updates the probability distribution $p(r)$ using a variant of Equation 9.8:
$$p(r) \xleftarrow{f,\hat{x},\hat{r}} \frac{p(f,\hat{x},\hat{r} \mid r)}{p(f,\hat{x},\hat{r})} p(r). \tag{9.30}$$
where the term $p(f,\hat{x},\hat{r} \mid r)$ is taken directly from the model.

- **Binary Update** When the robot moves from location $\hat{r}'$ to $\hat{r}$, as measured in the OCS, the distribution $p(r)$ is updated using a variant of Equation 9.9:
$$p(r) \xleftarrow{\hat{r},\hat{r}'} \int \frac{p(\hat{r},\hat{r}' \mid r,r')}{p(\hat{r},\hat{r}')} p(r')dr'. \tag{9.31}$$
where the term $p(\hat{r},\hat{r}' \mid r,r')$ is taken directly from the model.

Note that these two kinds of measurements have very different effects on $p(r)$. The detection of new landmarks tends to concentrate the distribution (decreasing the number of possible robot locations), while motion of the robot tends to 'smear' the distribution (increasing the number of possible robot locations). This is, however, not universally true – if the robot starts out with a uniform initial distribution, the binary update equation will tend to *decrease* the number of possible locations (at least up to a point). This is a result of the connectivity information encoded in the model – as the robot moves, Equation 9.31 will eliminate those locations that indicate that the robot has just passed through an obstacle.

Note that one of the most difficult problems for the localisation module is to ensure that the measurements being used to update the distribution $p(r)$ are statistically independent. As noted in Section 9.3.1, if the robot were to make multiple, identical measurements of the same landmark, the distribution may converge too quickly, leading to an incorrect assessment of the robot's true position. In practice, with the system described in this thesis, it is very difficult to guarantee such independence between measurements. Instead, a simple 'hack' is used – the localisation module keeps a list the measurements that have been made since the robot last moved, will reject a new measurement if it matches any of the measurements in this list. Thus, if the robot is stationary, the distribution $p(r)$ is updated *once* for each of the observed landmarks. While this hack appears to work, a more systematic approach would be desirable.

9.5.1 Implementation

The update equations described in the previous section are framed in terms of a continuous distribution $p(r)$ over the space of all possible robot locations. In reality, of course, the localisation module maintains a *discrete representation* of this distribution. Unfortunately, as I will show, this is a fundamentally difficult problem – maintaining an accurate representation of a continuous distribution either requires large amounts of memory, large amounts of time, or both. Consequently, I will describe two different representations – a *grid-based* representation and a *population-based* representation – and consider their relative merits.

A Grid-based Representation

In the grid-based representation, the robot's location space is divided into a large number of cells using a regular grid. Associated with each cell is a value $p(r)$ indicating the probability that the robot is in that cell. See, for example, Figure 9.4 – the dark cells in this figure indicate regions where the robot is likely to be found.

The key advantage of the grid-based approach is its simplicity – the distribution can be represented by a simple three-dimensional real-valued array. Unfortunately, the grid-based approach also has a number of disadvantages:

- It is expensive in its use of memory. Consider, for example, a grid in which cells measure 10cm by 10cm by 1°; if the environment measures 20m by 20m, the grid will contain over 14 *million* cells.

- It is expensive in its use of CPU time. Needless to say, updating a grid containing 14 million cells can be very time consuming.

To some extent, these problems can be overcome with careful programming – one can employ complex data structures for encoding the grid (such as quad-trees), and one can limit the update process to considering only those cells whose probability exceeds a threshold value.

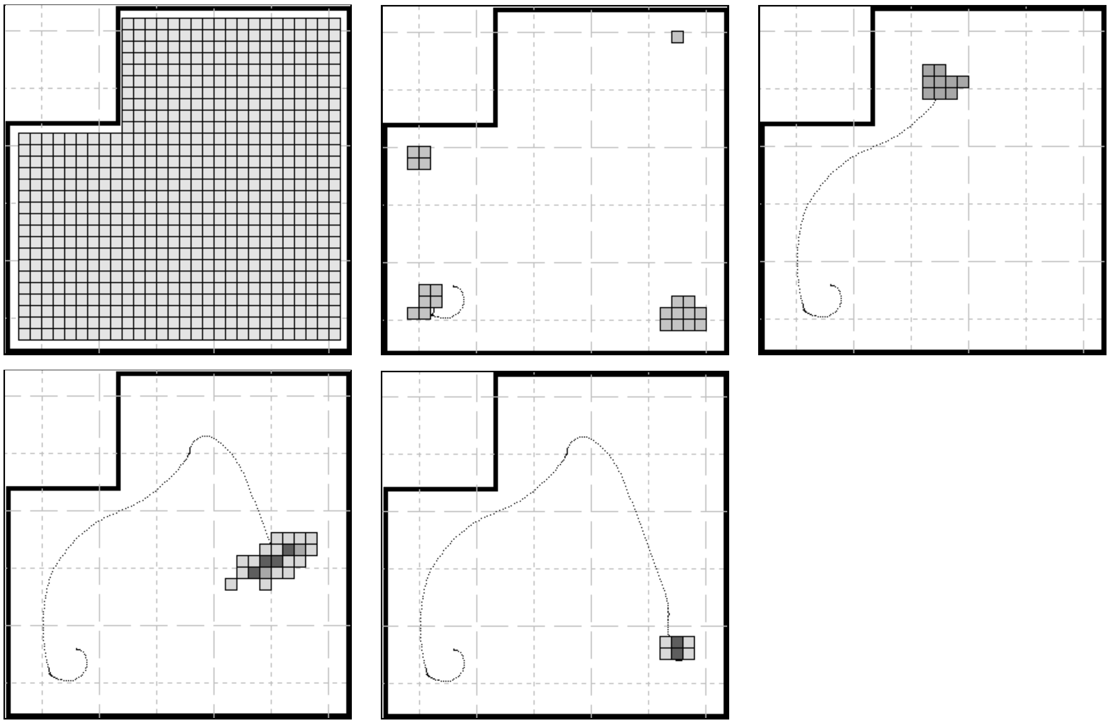


Figure 9.4: Evolution of the distribution $p(r)$ using a grid-based representation. The robot moves from its initial location (bottom left) to the goal location (bottom right). The dotted line shows the actual path of the robot.

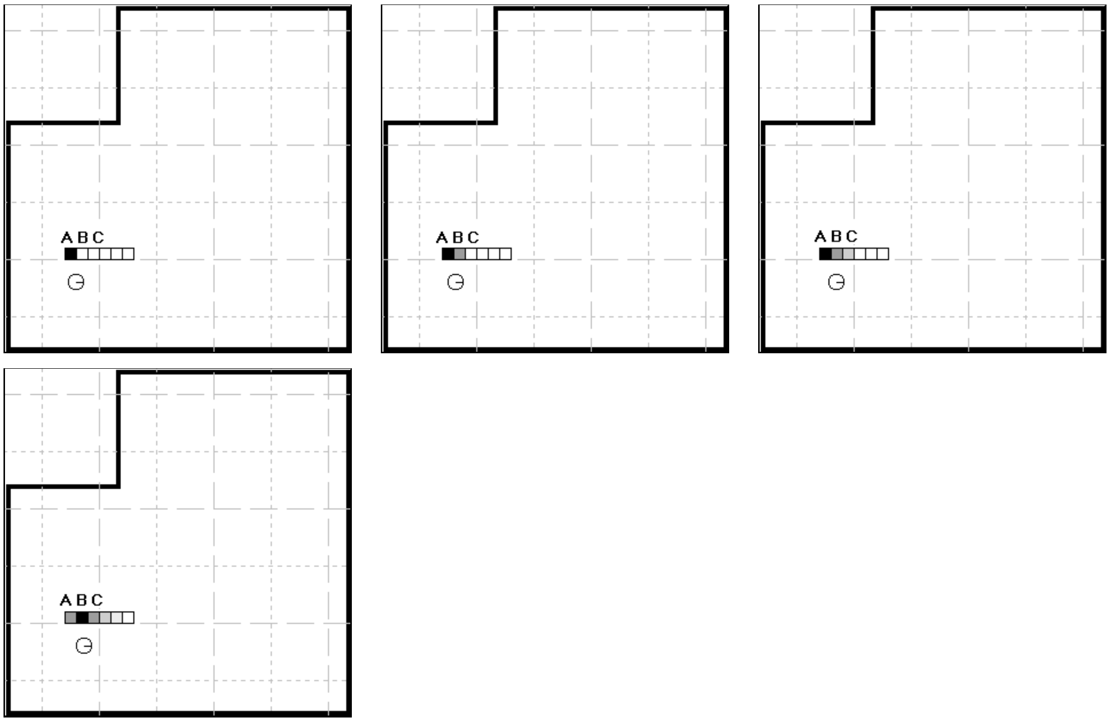


Figure 9.5: An example of the failure of the grid-based representation to capture small changes in the robot's location. The small circle indicates the robot's true location, which changes by only a few centimetres. The distribution $p(r)$, on the other hand, changes quite dramatically. Note that, for the sake of clarity, the grid has been vertically offset.

However, the grid-based representation has another problem, one that cannot be addressed through careful programming: it is not very good at capturing the effects of *changes* in the robot location. Consider, for example, the situation illustrated in Figure 9.5. Initially, it is known that the robot is in the cell marked A, and is not in either of the cells marked B or C. Thus we have:

$$p(A) = 1, p(B) = 0, p(C) = 0. \tag{9.32}$$

The robot now moves 1cm, as measured by odometry. While the robot is probably still in cell A, there is a finite probability that the robot has arrived in cell B (since it might have been sitting right on the boundary between these two cells). Thus we have:

$$p(A) = 0.9, p(B) = 0.1, p(C) = 0. \tag{9.33}$$

The robot now moves another 1cm. The probability that the robot is in cell B is now increased, but there is also a finite probability that the robot has arrived at cell C:

$$p(A) = 0.81, p(B) = 0.18, p(C) = 0.01. \tag{9.34}$$

This is, of course, absurd – after travelling only 2cm, there is a finite probability that the robot has arrived at cell C, about 20cm from where it first started! Note that this is not a consequence of odometric uncertainty; it is an artefact produced by the finite size of the grid cells. While one can reduce the impact of such artefacts by decreasing the cell size, the corresponding increase in the number of cells can quickly make the whole approach unmanageable. Alternatively, instead of updating the grid whenever the robot moves, one can update the grid whenever the robot moves some minimum distance (say 10cm). Unfortunately, while this avoids the kind of artefacts shown in Figure 9.5, it leads to very coarse modelling of odometric uncertainty.

A Population-based Representation

The population-based representation is somewhat more complex than the grid-based representation, and has very different characteristics. In this approach, the localisation module maintains a *population* of candidate locations, each of which has a probability assigned to it. However, unlike the cells in the grid-based approach, these candidates are allowed to *move*.

The population, which is both fixed and finite, is subjected to three processes:

- **Unary update** When a new landmark is detected, the probability of each candidate is updated using Equation 9.30:

$$p(r) \xleftarrow{f,\hat{x},\hat{r}} \frac{p(f,\hat{x},\hat{r} \mid r)}{p(f,\hat{x},\hat{r})} p(r). \tag{9.35}$$

- **Binary update** When the robot moves, each candidate is translated according to the rule:

$$r' \xleftarrow{\hat{r},\hat{r}'} \Gamma_{r\hat{r}} \hat{r}' + \epsilon. \tag{9.36}$$

  The first term in this expression $\Gamma_{r\hat{r}} \hat{r}'$ describes the *expected* change in the robot's location, as determined using odometry. The second term ϵ is a small perturbation whose value is chosen *at random* (specifically, ϵ is drawn from a uniform random distribution whose variance is determined experimentally). The probability of the candidate is then updated using a modified form of Equation 9.31:

$$p(r') \xleftarrow{\hat{r},\hat{r}'} \frac{p(\hat{r}',\hat{r} \mid r', r)}{p(\hat{r}',\hat{r})} p(r) \tag{9.37}$$

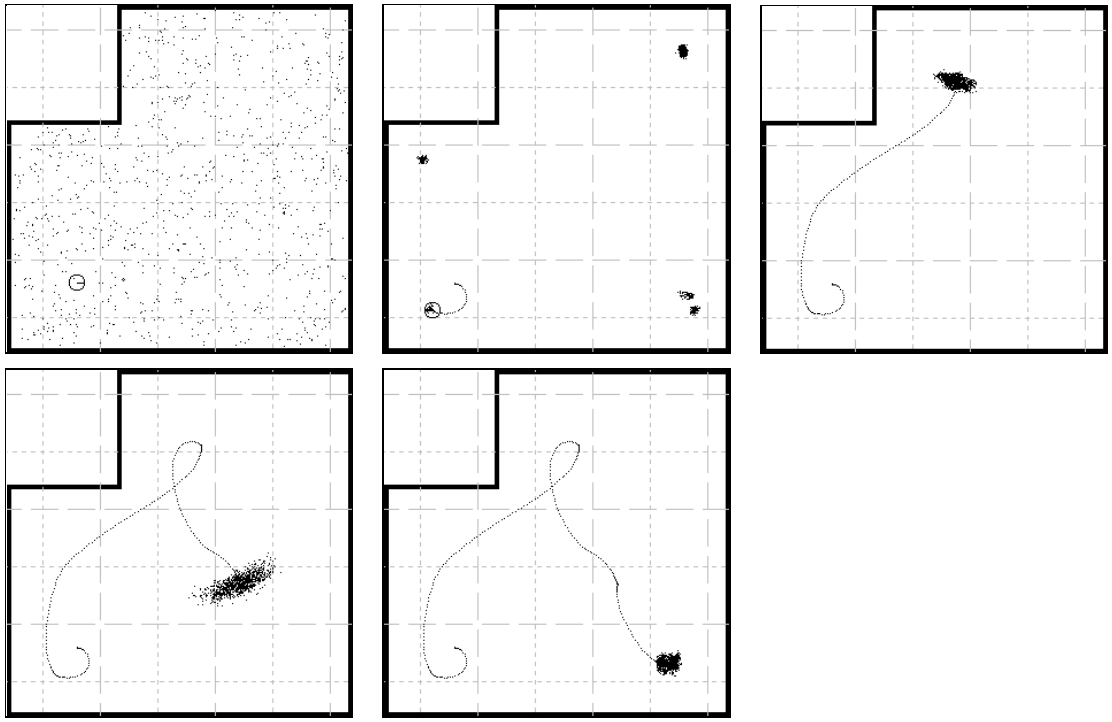


Figure 9.6: Evolution of the distribution $p(r)$ using a population-based representation. Each dot represents one possible robot location (i.e. one candidate in the population). The robot moves from its initial location (bottom left) to the goal location (bottom right). The dotted line shows the actual path of the robot.

Consequently, the path taken by a particular candidate represents one *possible* path the robot might have followed. The probability that the robot *actually* followed this path is given by $p(r)$. Note that, since odometric data may arrive at up to 10Hz, this binary update can be a relatively expensive process.

- **Reproduction** The population is subject to a periodic process of selection and reproduction. Candidates with a very low probability are removed from the population and replaced with a clone of the candidate with the highest probability. The probability assigned to the original candidate is divided equally between itself and its clone.

The net effect of these processes is that, over time, the population tends to cluster around the more probable regions of the location space. Consequently, the less uncertainty there is in the robot's location, the more accurately the population approximates the ideal, continuous distribution. As a concrete example, consider Figure 9.6. Each dot in this figure corresponds to a candidate location; the darker dots indicate the more probable candidates. Note how the candidates form clusters around the most probable regions of the location space, and how those clusters move with the robot.

Overall, the population-based approach can be thought of as a kind of genetic algorithm [114, 100] with a dash of Monte-Carlo simulation thrown in [255]. This approach has two key advantages:

- It captures changes in the robot's location very well, since the candidates themselves are allowed to move.

- It is relatively inexpensive in its use of memory. The population shown in Figure 9.6, for example, contains only 1000 candidates.

Unfortunately, this population-based approach also has a big drawback – there is no guarantee that the population contains a candidate that approximates the *true* location. This is particularly true if the population size is small and the initial location is unknown. In this case, the robot may be mis-localised; that is, there will be a substantial disagreement between where the robot thinks it is, and where it really is. Consider, for example, the situation depicted in Figure 9.7. The initial location of the robot is unknown, and is represented by a population containing 1000 randomly selected candidates. It so happens that in this population, there is no candidate that comes even close to approximating the true location. Consequently, as the robot moves through the environment, a mis-match soon develops. At first, the robot merely concludes that it is somewhere where it is not; eventually, however, the robot eliminates all of the candidates in the population and is forced to conclude that it is *nowhere*. Fortunately, this situation is easy to handle – the population is re-initialised and the whole process starts again. More problematic is the case when the robot concludes that it has arrived at the goal when it is, in fact, somewhere that merely *looks* like the goal. Naturally, this particular problem can be alleviated by increasing the population size; however, as with the grid-based approach, this can quickly become an expensive process. We will consider the problem of mis-localisation in an experimental context in Section 9.7.

Choosing a Representation

All of the results described in this chapter were obtained using a population-based representation. This representation was chosen over the grid-based representation for a number of reasons: it is somewhat less expensive, and it is better at capturing small changes in the robot's location. Mostly however, the population-based

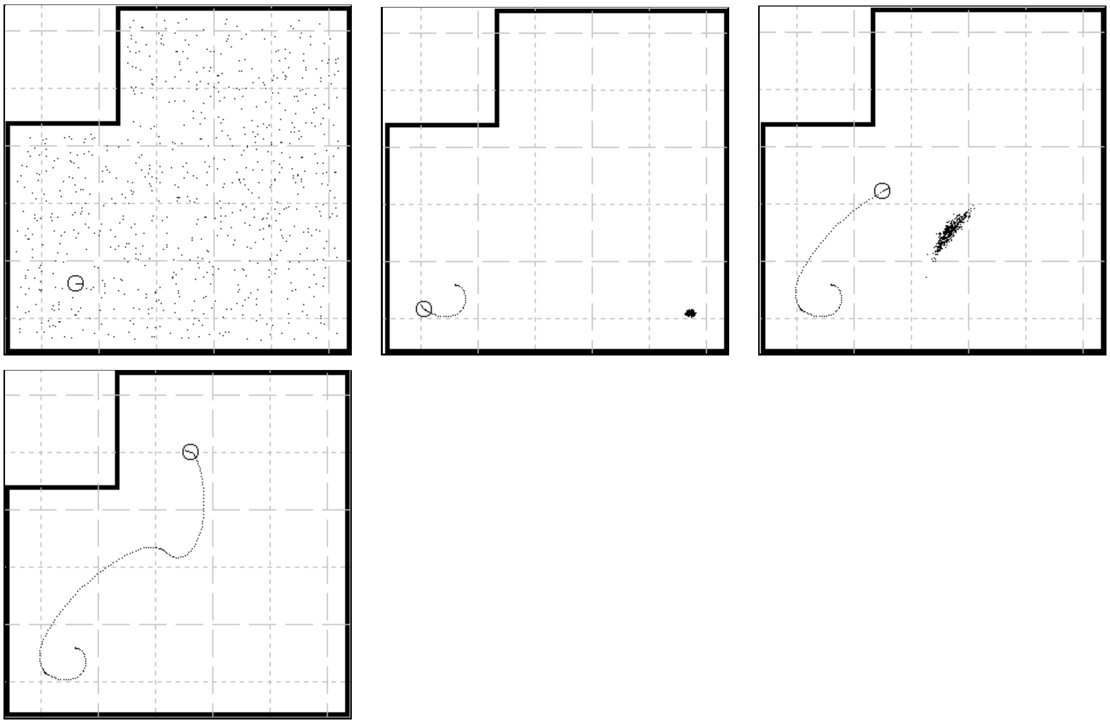


Figure 9.7: An example of a localisation failure using the population-based representation. The initial population was too small to capture to correct initial location, with the result that the robot is first mis-localised, and then lost entirely.

representation was chosen because it facilitates the use of the *persistent* navigation strategy. This strategy is described in the next section.

9.6 The Planning Module

In its simplest form, the planning problem can be described as follows: given that the robot's location is known, find a series of actions that will take the robot from its initial location to the goal location. With the probabilistic localisation approach, however, planning becomes much more complicated – one does not know the location of the robot, one knows only the probability distribution $p(r)$. Consequently, with this approach, the planning problem must be stated somewhat differently: given that the distribution $p(r)$ is known, find a series of actions that will transform the initial distribution (which may be uniform) into a distribution that is peaked around the goal. To put this another way: one must find a series of actions such that the robot will not only reach the goal, it will *know* that it has reached the goal.

It is possible, in principle, to find an optimal solution to this problem. The robot's *state* can be described in terms of the distribution $p(r)$. The space of such states can then be searched to find a series of actions that connects the robot's initial state to its goal state. In reality, of course, it is extremely difficult to perform such a search. Since each point in the search space corresponds to a *continuous* distribution, the search space is extremely large (even for the most trivial examples). Consequently, this approach is impractical for realistic environments.[3]

[3] As a number of authors have observed [130, 140], this planning problem can be formulated as a *Partially Observable Markov Decision Process* (POMDP). Within the POMDP framework, there are efficient algorithms for finding an optimal *policies* to guide the robot's actions. Unfortunately, none of these algorithms comes close to being computationally tractable for real environments.

As an alternative, the planning module makes use a number of sub-optimal *navigation strategies*. Navigation strategies represent a partial solution to the planning problem – they will enable the robot to reach the goal in a reasonable amount of time, on most occasions, and in most environments.

Navigation strategies are composed of three elements: a *plan*, a *selection rule* and a *termination rule*:

- A *plan* is a function that specifies, for every location r, the action a the robot *should* perform when it is at that location. We write this as:

$$a = \text{Plan}(r). \tag{9.38}$$

  We will look at how such plans may be constructed in Section 9.6.

- The robot can make use of this plan, even when its location is uncertain, by augmenting it with a *selection rule*. In its simplest form, the selection rule picks out one possible location and directs the robot to perform the corresponding action in the plan:

$$a = \text{Plan}(\text{Sel}[p(r)]) \tag{9.39}$$

  More complex selection rules are also possible.

- The robot applies this procedure until the *termination rule* is satisfied, at which point it will conclude that it has reached the goal, and stop.

As a concrete example, consider the so-called *most probable* strategy. With this strategy, the robot simply picks the most probable location from the distribution $p(r)$, and executes the corresponding action in the plan; this procedure continues until the most probable location is the goal. Figure 9.1 shows the path taken by a simulated robot using the most probable strategy. Initially, the robot's location is unknown, and the distribution $p(r)$ is uniform. Since all locations are equally likely, the robot simply picks one at random and performs the corresponding action in the plan. Unfortunately, the robot's guess is incorrect, with the result that the robot initially moves *away* from the goal. Soon, however, the detection of a series of landmarks narrows down the set of possible locations, and the distribution $p(r)$ collapses into single peak. At this point, since the robot's location is well determined, the robot is able to head directly for the goal. Note that the goal is defined as a region rather than a point; given the discrete nature of the robot's actions, a robot given a point goal will tend to circle *around* the goal, rather than actually reaching it.

Navigation strategies by no means represent the *only* solution to the planning problem defined above. They do, however, offer a simple yet effective mechanism for solving this problem. In the following sections, I will consider, in detail, three different navigation strategies, all of which employ the same plan, but differ in their selection and termination rules. First, however, we must look at how the plan is constructed.

9.6.1 Constructing the Plan

The plan is constructed by considering the *cost* of reaching the goal from any given location. Usually, cost is a measure of the time required to reach the goal or the distance the robot must travel. Strictly speaking, this cost must be an *expected* cost, since the outcome of actions is always somewhat uncertain.

Let $\mathrm{Cost}(r)$ denote the expected cost of reaching the goal from location r. If we require that, at any given location, the robot always performs the action that minimises this cost, we can immediately write down an expression for the cost function: [4]

$$\text{Cost}(r) = \min_a \int_{r'} p(r' \mid r, a)\text{Cost}(r')dr' + \text{Cost}(a). \tag{9.40}$$

where

- $p(r' \mid r, a)$ describes the probability that the robot will arrive at location r', given that it is at location r and performs action a. This term expresses the uncertainty associated with the outcome of actions, and is part of the global model.

- $\text{Cost}(a)$ describes the *atomic* cost associated with the action a. In this thesis, we assume that this cost is identical for all actions.

The cost function is also subject to the boundary condition:

$$\text{Cost}(\text{goal}) = 0; \tag{9.41}$$

that is, the cost of reaching the goal, from the goal, is zero.

Note that the cost function defined through Equations 9.40 and 9.41 has an extremely useful property – it has no *local* minima. It does, however, have a *global* minimum – the one defined by the boundary condition, at the goal. The proof of this is straight-forward. The integral term in Equation 9.40 computes a weighted average of the cost of all of the locations in the neighbourhood of r that can be reached by an action a. This weighted average must, by definition, return a value that is greater than or equal to the *minimum* cost for same neighbourhood. Therefore, if the atomic

[4]The reader may recognise this as a result from Markov Decision Process Theory [86], or, alternatively a form of Dynamic Programming [26]. It also has analogues in the Distance Transform [127], Gradient Fields [192] and D* [231] approaches.

cost for an action $\mathrm{Cost}(a)$ is positive, the cost of any location r, given that the robot performs action a, must be *greater than* the cost of any of its neighbours. The only exception is the goal; the boundary condition defined in Equation 9.41 ensures that this location is a global minimum.

From the definition of the cost function, if follows that the plan function is given by:

$$\mathrm{Plan}(r) = \arg\min_a \int_{r'} p(r' \mid r, a)\mathrm{Cost}(r') + \mathrm{Cost}(a). \tag{9.42}$$

that is, the robot always performs the action that minimises the expected cost. In effect, the robot is performing a kind of gradient descent, always seeking the location with the lowest cost. Since the cost function has only one minimum, defined by the boundary condition, this procedure guarantees that the robot will eventually arrive at the goal.

As a concrete example, consider the situation depicted in Figure 9.8. Figure 9.8(a) shows the cost function for a simulated environment (the goal is at the bottom right). Note the absence of local minima. Figure 9.8(b) shows the corresponding plan function, with the arrows indicating the robot's preferred action. Clearly, if the robot follows this plan, it will always arrive at the goal.

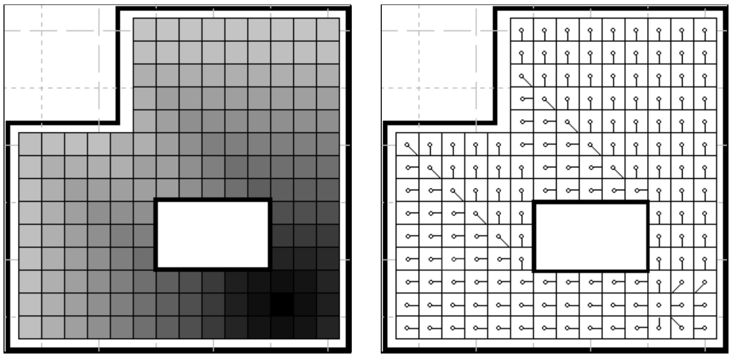
In practice, the cost function defined in Equation 9.40 can be solved numerically using a simple value iteration scheme. The procedure is as follows:

1. Let $\mathrm{Cost}_i(r)$ denote the value of the cost function on the i^{th} iteration.

2. Set $\mathrm{Cost}_i(\mathrm{goal}) = 0$ for all i (this is the boundary condition).

3. Set $\mathrm{Cost}_0(r) = \infty$ for all $r \neq \mathrm{goal}$.

4. Compute the value of $\mathrm{Cost}_{i+1}(r)$ from $\mathrm{Cost}_i(r)$ using a variant of Equation



(a) (b)

Figure 9.8: (a) Cost function for a simple environment (the goal is at the bottom right); darker cells have a lower cost. (b) The corresponding plan function; the planned action for each location is indicated by the arrows.

9.40.

$$\mathrm{Cost}_{i+1}(r) = \min_a \sum_{r'} p(r' \mid r, a)\mathrm{Cost}_i(r') + \mathrm{Cost}(a) \tag{9.43}$$

In this expression, the integral has been replaced with a summation. If one is using a grid-based representation for the model, this summation is carried out over all the cells in the grid; if one is using a graph-based representation, the summation is over all nodes in the graph.

5. Repeat step 4 until the total difference between cost functions on successive iterations falls below some arbitrary cut-off value.

The value of $\mathrm{Cost}_i(r)$ will converge to the true value in the limit $i \to \infty$. The plan function can then be computed in a straight-forward manner using Equation 9.42.

9.6.2 Navigation Strategies

The planning module is capable of implementing three different navigation strategies: the *most probable strategy*, the *proportional voting strategy* and the *persistent strategy*. The most probable and proportional voting strategies have been chosen because of their simplicity, and because they are analogous to similar techniques described in previous work on probabilistic navigation [130, 140]. The persistent strategy is novel, and is designed to address some of the observed limitations of the first two strategies.

The Most Probable Strategy

Of the three navigation strategies, the most probable strategy is the simplest: it directs the robot to perform the action corresponding to the most probable location.

The robot continues to perform this action until the most probable location indicates that the robot is at the goal. The strategy is defined formally as follows:

- Selection rule: select the location r_s such that:

  $$r_s = \arg\max_r p(r); \tag{9.44}$$

  and perform the action given by

  $$\hat{a} = \Gamma_{\hat{r}r_s} \mathrm{Plan}(r_s). \tag{9.45}$$

  Note that the inverse transform $\Gamma_{\hat{r}r_s}$ is used to determine the required action with respect to the OCS.

- Termination rule: Terminate when $r_s = \text{goal}$.

The Proportional Voting Strategy

The proportional voting strategy is somewhat more democratic than the most probable strategy. Each location is allowed to 'vote' for an action in proportion to the probability that that location is correct. The robot then executes the action with the most votes. The proportional voting strategy is defined formally as follows:

- Selection rule: Let the number of votes for an action $\hat{a}$ be defined as follows:

  $$\mathrm{Votes}(\hat{a}) = \int p(r)\delta[\hat{a}, \Gamma_{\hat{r}r}\mathrm{Plan}(r)]dr \tag{9.46}$$

  where the delta function $\delta(\cdot,\cdot)$ returns 1 if its arguments are equal and zero otherwise. The robot performs the action with highest number of votes, that is, the action given by

  $$\hat{a} = \arg\max_{\hat{a}'} \mathrm{Votes}(\hat{a}'). \tag{9.47}$$

- Termination rule: terminate when $\hat{a} = \text{stop}$.

The rationale behind this selection rule is that it allows all possible locations to have a say in what the robot should do, but gives more weight to those locations with a high probability.

Persistent Strategy

The persistent strategy is designed to address two problems that are observed experimentally with the first two strategies.

- Both strategies are prone to oscillation. This occurs when the robot rapidly switches back and forth between two actions, as subtle changes in the shape of the distribution $p(r)$ result in different actions being selected. While the robot will generally extract itself from such situations eventually, it can waste a great deal of time in doing so.

- There is a high probability that the robot will reach a place that looks like, but is not, the goal. Consider, for example, the most probable strategy. This strategy will terminate as soon as the most probable location indicates that the robot is that the goal. However, the probability that this location is correct, and hence the probability the robot is *actually* at the goal, could be anywhere between zero and one.

Both of these problems are largely eliminated by the persistent strategy.

The persistent strategy works in conjunction with the population-based representation for $p(r)$ described in Section 9.5.1 (at this point, it is unclear whether the strategy can be applied to other representations). Under this strategy, the robot selects a single candidate from the population of candidate locations and performs the corresponding action in the plan. The robot *persists* with the selected candidate until one of two conditions is met: either the selected candidate indicates that the robot has reached the goal, or the candidate is eliminated from the population (i.e. its probability falls below some arbitrary cut-off value). In either case, the robot is free to select a new candidate. This procedure terminates only when all remaining candidates in the population indicate that the robot is at the goal.

Under the persistent strategy, the robot is effectively *testing* each of the candidates in the population. There are can only be two outcomes of such a test: either the candidate will indicate that the robot has arrived at the goal (which may or may not be true), or the candidate will be eliminated. The robot can maximise the chances of eliminating a candidate, and hence minimise the chances of reaching a *false* goal, by selecting the candidate that is *most distant* from the goal. Consequently, when selecting a new candidate, the persistent strategy (which should perhaps be called the persistent *pessimist* strategy) always chooses the candidate that maximises the cost function $\mathrm{Cost}(r)$.

Note that, roughly speaking, it is the *persistent* nature of this strategy that ensures that oscillations are all but eliminated, and the *pessimistic* nature of this strategy that yields a high probability that the robot will arrive at the goal. These effects are quantified in Section 9.7.

The persistent strategy is defined formally as follows:

- Selection rule: select a candidate r_s such that:

$$r_s = \arg\max_{r'} \mathrm{Cost}(r'); \tag{9.48}$$

  and perform the corresponding action in the plan:

$$\hat{a} = \Gamma_{\hat{r}r_s}\mathrm{Plan}(r_s) \tag{9.49}$$

Persist with the selected candidate until r_s = goal or the candidate is eliminated from the population.

- Termination rule: stop when all remaining candidates lie within some arbitrarily small distance from the goal.

Choosing a Strategy

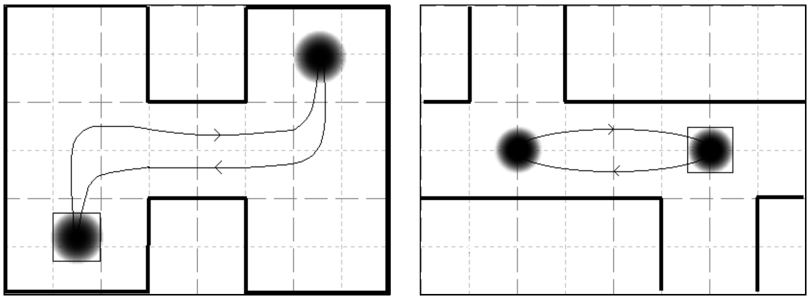
In Section 9.7, we will show that the persistent strategy significantly out-performs the most probable and proportional voting strategies, both in terms of suppressing oscillations and maximising the probability that the robot will reach the goal. There are, however, some situations that cannot be solved using this strategy (indeed, there are some situations that cannot be solved using *any* strategy). Some examples of such situations are considered in the following section.

9.6.3 A Note on Pathological Environments

Consider the simple environment shown in Figure 9.9(a) – this is an example of a pathological environment. Clearly, since the environment is rotationally symmetric, there is no method by which a robot whose location is initially unknown can unambiguously determine its location. Hence, there is no strategy, of any kind, by which the robot can both reach the goal and *know* that it has reached the goal. The three navigation strategies respond to this environment in different ways.

- The most probable strategy will quickly terminate, but the probability that the robot is at the goal will be no more than 50%.

- The proportional voting strategy will also terminate; the probability that the robot is at the goal will be no more than 50%.



(a) (b)

Figure 9.9: (a) Pathological environment. Under the persistent strategy, the robot will shuttle back-and-forth on the indicated path. (b) Pathological situation. While the environment as a whole is not pathological, the particular population distribution shown in the figure is.

- The persistent strategy will never terminate, since there will always be some transform that needs to be tested. The robot will shuttle back-and-forth between the two locations that could be the goal.

This is, of course, an extreme example — real environments are not usually constructed in such a fashion (with the possible exception of major international airports).

A similar effect can occur, however, if the environment contains rotationally symmetric *regions* and if the goal is contained in one of these regions. Consider, for example, the section of corridor depicted in Figure 9.9(b). For most configurations of the location distribution, this environment does not pose a problem. However, for the particular configuration shown in Figure 9.9(b), this environment is pathological — the robot will either arrive at the wrong goal (with probability 50%) or will shuttle back and forth between the two doorways. This example, which can occur quite frequently in real environments, highlights the need for navigation strategies that are more sophisticated than those described here. Such strategies are the subject of ongoing research.

9.7 Experiments

In this section, the overall performance of the navigation system is evaluated for a range of environments, both simulated and real. In addition, the individual performance of the three navigation strategies described in Section 9.6 is compared.

9.7.1 Protocol

The overall performance of the system is evaluated by conducting repeated trials. In each trial, the robot is placed at some initial location (which may or may not be known to the robot) and is allowed to proceed to the goal (or perhaps to a place that it thinks is the goal). Two statistical performance measures are used:

- The total failure rate, which is defined as the fraction of trials in which the robot fails to reach the goal;
- The mean time per trial, which is defined as the average time taken to complete each trial.

Clearly, the navigation system should aim to minimise both of these measures.

In addition to measuring the performance of the system as a whole, we are also interested in comparing the individual performances of the three different navigation strategies. To do this, we need to distinguish between effects that can be attributed to the localisation system and effects that must be attributed to the navigation strategy alone. That is, in those cases in which the robot fails to reach the goal, we would like to know whether this failure was due to a failure of the localisation system, or a failure of the navigation strategy. The tool for making this distinction is called a *confusion matrix* and takes the following form.

	$\bar{g}$	$\neg\bar{g}$
g	$\mathrm{p}(g \wedge \bar{g})$	$\mathrm{p}(g \wedge \neg\bar{g})$
$\neg g$	$\mathrm{p}(\neg g \wedge \bar{g})$	$\mathrm{p}(\neg g \wedge \neg\bar{g})$

In this matrix, the proposition g is true if the robot reaches the goal. Similarly, the proposition $\bar{g}$ is true if the robot *believes* that it has reached the goal, i.e. if the probability $p(\text{goal})$ exceeds 0.8 (finding a good definition for what the robot *believes* can be tricky, but this definition seems as good as any). The first row of the confusion matrix gives two values:

- $\mathrm{p}(g \wedge \bar{g})$ is the fraction of trials in which the robot reached the goal and believed that it had reached the goal.

- $\mathrm{p}(g \wedge \neg\bar{g})$ is the fraction of trials in which the robot reached the goal but did not believe that it had reached the goal.

The second row of the confusion matrix gives the complementary values:

- $\mathrm{p}(\neg g \wedge \bar{g})$ is the fraction of trials in which the robot did not reach the goal but believed that it had reached the goal.

- $\mathrm{p}(\neg g \wedge \neg\bar{g})$ is the fraction of trials in which the robot did not reach the goal and did not believe that it had reached the goal.

Clearly, if the robot's localisation system is perfect, the off-diagonal elements of the confusion matrix — which indicate a disagreement between where the robot actually is and where the robot thinks it is — should be zero. Similarly, if the navigation strategy is perfect, the on-diagonal element $\mathrm{p}(\neg g \wedge \neg\bar{g})$ — which can be interpreted as the fraction of trials in which the robot failed to reach the goal, even though the localisation system worked flawlessly — should be zero. We therefore define the following additional pair of performance measures:

- The localisation failure rate, given by $\mathrm{p}(\neg g \wedge \bar{g})$.

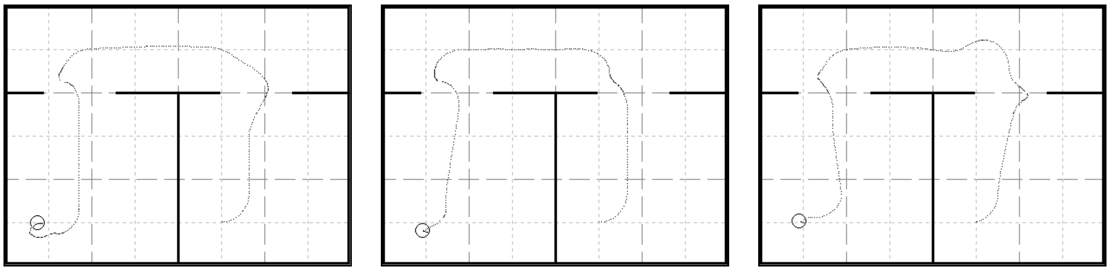
- The strategy failure rate, given by $\mathrm{p}(\neg g \wedge \neg\bar{g})$.

Note that we count as failures only those trials in which the robot does not arrive at the goal. Trials in which the robot arrives at the goal, but does not believe that it has arrived at the goal, are not counted as failures. Such outcomes are quite rare, of course.

9.7.2 A Simulated Environment with a Known Initial Location

Consider the series of images shown in Figure 9.10. Each of these images depicts a path taken by the robot through a simulated environment using one of the navigation strategies described in Section 9.6. In each case, the initial location was known to the robot. This is the simplest possible example – it corresponds to the navigation task as it is typically framed – and as we might expect, all paths are more-or-less identical.

In order to accurately assess the performance of the navigation system, many trials of this experiment were conducted. In each trial, the robot's initial location was chosen at random, and this location was known to the robot. The results are presented in Table 9.1. Clearly, with this simple example, the system performs very well, exhibiting zero failures in 184 trials. The only difference between the three navigation strategies is in the average time requires for each trial. Here, the persistent strategy seems to have a slight advantage over the other two strategies – an advantage which can perhaps be attributed to the suppression of oscillations, as noted in Section 9.6.2.



(a) (b) (c)

Figure 9.10: Results for the simulated environment with known location. The robot starts at the bottom right; the goal is at the bottom left. (a) Path taken by the robot using the most probable navigation strategy. (b) Proportional voting strategy. (c) Persistent strategy.

	Most Probable	Prop. Voting	Persistent
Trials	61	65	58
Mean time per trial (sec)	57	65	50
Total failure rate (%)	0	0	0
Localisation failure rate (%)	0	0	0
Strategy failure rate (%)	0	0	0

Table 9.1: Results for a simulated environment with known location.

9.7.3 A Simulated Environment with an Unknown Initial Location

Consider the series of images shown in Figure 9.11. Each of these images depicts a path taken by the robot through a simulated environment using one of the navigation strategies described in Section 9.6. This time, however, the initial location was *unknown* to the robot. This is a much more interesting example: it corresponds to the navigation task as it is framed in the Introduction – how does a robot with an *unknown* initial location go about reaching the goal? Note how the three paths are quite different. Under the most probable strategy, the robot traverses a convoluted path before eventually moving directly to the goal. Under the proportional voting strategy, the robot entirely fails to reach the goal – it arrives instead at a place that looks like, but is not, the goal. Finally, under the persistent strategy, the robot traverses a more-or-less direct path to the goal.

While the paths shown in Figure 9.11 are typical, they naturally represent only one possible outcome of such an experiment. The results for repeated trials of this experiment are given in Table 9.2, which lists both the mean time per trial and the overall failure rate. For both measures, a 95% confidence interval has been calculated (shown in square brackets). Note that the relatively large interval on the mean time per trial mostly reflects the variation in the robot's initial location — trials will take less time when the robot starts near the goal.

Inspecting this table, it is apparent that a significant fraction of the failures can be attributed to the localisation system. There are a number of possible causes for such a failure:

- The model may be inaccurate.

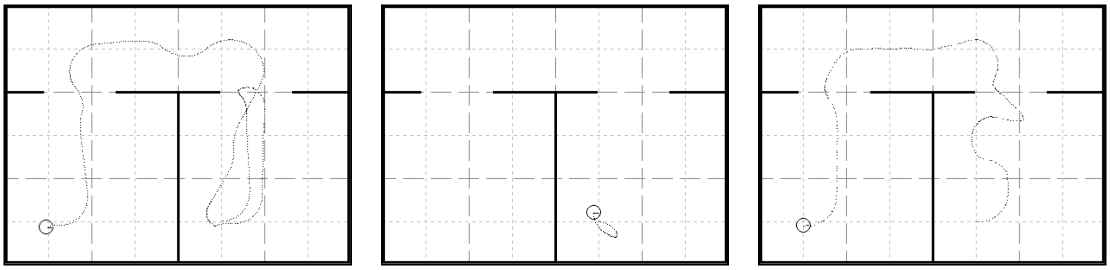


Figure 9.11: Results for the simulated environment with unknown location. The robot starts at the bottom right; the goal is at the bottom left. (a) Path taken by the robot using the most probable navigation strategy. (b) Proportional voting strategy. Note that the robot failed to reach the goal in this instance. (c) Persistent strategy.

	Most Probable	Prop. Voting	Persistent
Total trials	234	238	244
Failures	58	33	13
Mean time per trial (sec)	63 [±23]	69 [±21]	100 [±49]
Total failure rate (%)	25 [20, 32]	14 [10, 20]	5 [3, 10]
Localisation failure rate (%)	3	3	3
Strategy failure rate (%)	22	11	2

Table 9.2: Results for the simulated environment with unknown location.

- The landmark detection agent may produce a spurious match.
- The population size may not be large enough to capture the correct initial transform.

Given that this is a simulation, the first two causes can presumably be discounted, which leads to the conclusion that localisation failures are most probably due to an inadequate population size (in this experiment a population size of 1000 was used).

The remainder of the failures in Table 9.2 must be attributed to failures of the navigation strategy. Comparing the three strategies, it is immediately apparent that the persistent strategy has a much lower failure rate than the other two. This difference can be further highlighted by breaking down the results to reveal the effect of initial location on the strategy failure rate. Consider Figure 9.12, which shows the failure rate for a number of initial locations superimposed on a map of the environment. From this figure, it is quite clear that while the average failure rates for the most probable and proportional voting strategies are 22% and 11% respectively, the failure rates for some initial locations are much higher. These failures are a direct result of the termination rules for both these strategies, which makes them prone to stopping at places that look like, but are not, the goal. As noted in Section 9.6.2, the most probable strategy will terminate as soon as the most probable location is the goal location, regardless of how probable this is. Similarly, the proportional voting strategy will terminate as soon as a weighted majority of transforms indicates that the robot is at the goal, i.e. when there is only a 50% chance that the robot is actually at the goal.

The failure rate for the persistent strategy, on the other hand, is relatively constant for all initial locations. This is to be expected, since this strategy does not suffer from the problem of reaching false goals. The question might then be asked —

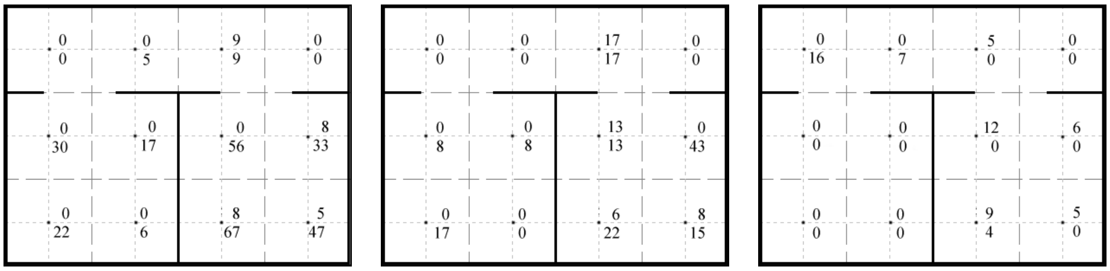
under what circumstances does the persistent strategy fail? There are two possible causes for failures of the persistent strategy:

- The robot may become 'stuck' behind some obstacle. That is, the robot may attempt to perform an action that would require it to move through an extended obstacle, such as a wall. Since the local navigation layer will refuse to carry out such actions, the robot will stop, and the navigation strategy will be deemed to have failed.

- The environment may contain regions that have either rotational or translational symmetries. In this case, it is possible for the population of transforms to assume a pathological configuration which traps the robot within the symmetric region of the environment. This scenario was dealt with in Section 9.6.3.

As it happens, all of the failures seen in this experiment were a result of the robot becoming stuck behind some obstacle.

Looking now at the mean time per trial, it can be seen that the persistent strategy takes somewhat longer on average than the other strategies. While it is tempting to attribute this difference to the pessimistic nature of the persistent strategy — it will always select transforms that indicate that the robot is far from the goal — a closer analysis of the data reveals that this is not the case. Consider the strategy failure rates shown in Figure 9.12. For the first two strategies, the failure rate is highest for those locations that are most distant from the goal. That is, for these locations, there is a high probability that the robot will find a *nearby* false goal. As a result, the mean time for the first two strategies is lower than would be expected if the robot was reaching the goal more often. Thus, the higher mean time exhibited



(a) (b) (c)

Figure 9.12: (a) Results for the simulated environment with unknown location using the 'most probable' navigation strategy. The map shows the localisation and strategy failure rates (expressed as percentages) for a range of initial locations. (b) Proportional voting strategy. (c) Persistent strategy.

by the persistent strategy can be attributed to the fact that with this strategy, the robot is reaching the goal more often.

9.7.4 A Real Environment with an Unknown Initial Location

Figure 9.13 shows a fragment of the environment that J Edgar normally inhabits. This is an ordinary office environment – made up of corridors, intersections, doorways and rooms – that has not been modified in any way for the robot. For the sake of both safety and simplicity, the robot's motion is restricted to the corridors and intersections of this environment, which tend to be less cluttered than the rooms (the robot is prevented from entering rooms by 'tweaking' the weights in the local navigation module until the robot refuses to pass through doorways).

Consider now Figure 9.14, which shows the results for a typical navigation trial using the persistent strategy. In this trial, the robot's initial location was chosen at random and this location was *unknown* to the robot. The left column in this Figure 9.14 shows the local occupancy map at a number of different points in time. The right column shows the corresponding population of candidate locations, with each dot representing one estimate of the robot's location. Looking at this sequence of images, it is clear that the detection of a number of landmarks quickly reduces the initial, uniformly distributed population to a population containing three distinct clusters. Further reduction, however, takes much longer. The robot has to travel about 10m before it can reduce the three clusters to just one cluster. At this point, the robot moves directly to the goal.

The results for 36 such trials are summarised in Table 9.3. In each trial, the robot's initial location was chosen at random and this location was *unknown* to the

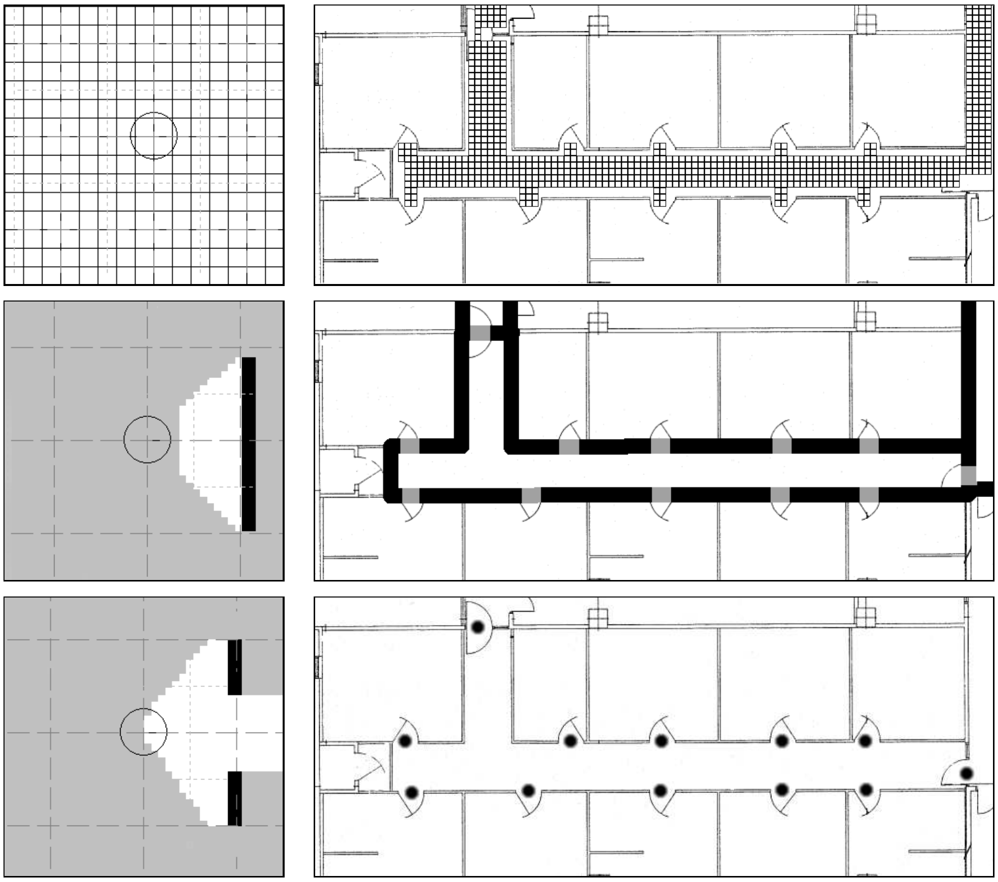


Figure 9.13: Model for a fragment of the real environment. Top row: free-space map — allowed locations are indicated by the open grid cells. Middle row: distribution of wall landmarks. The probability of observing a wall type landmark is indicated by the shading; hence the doorways (which may be either open or closed) are a lighter shade than the fixed walls. Bottom Row: distribution of doorway landmarks.

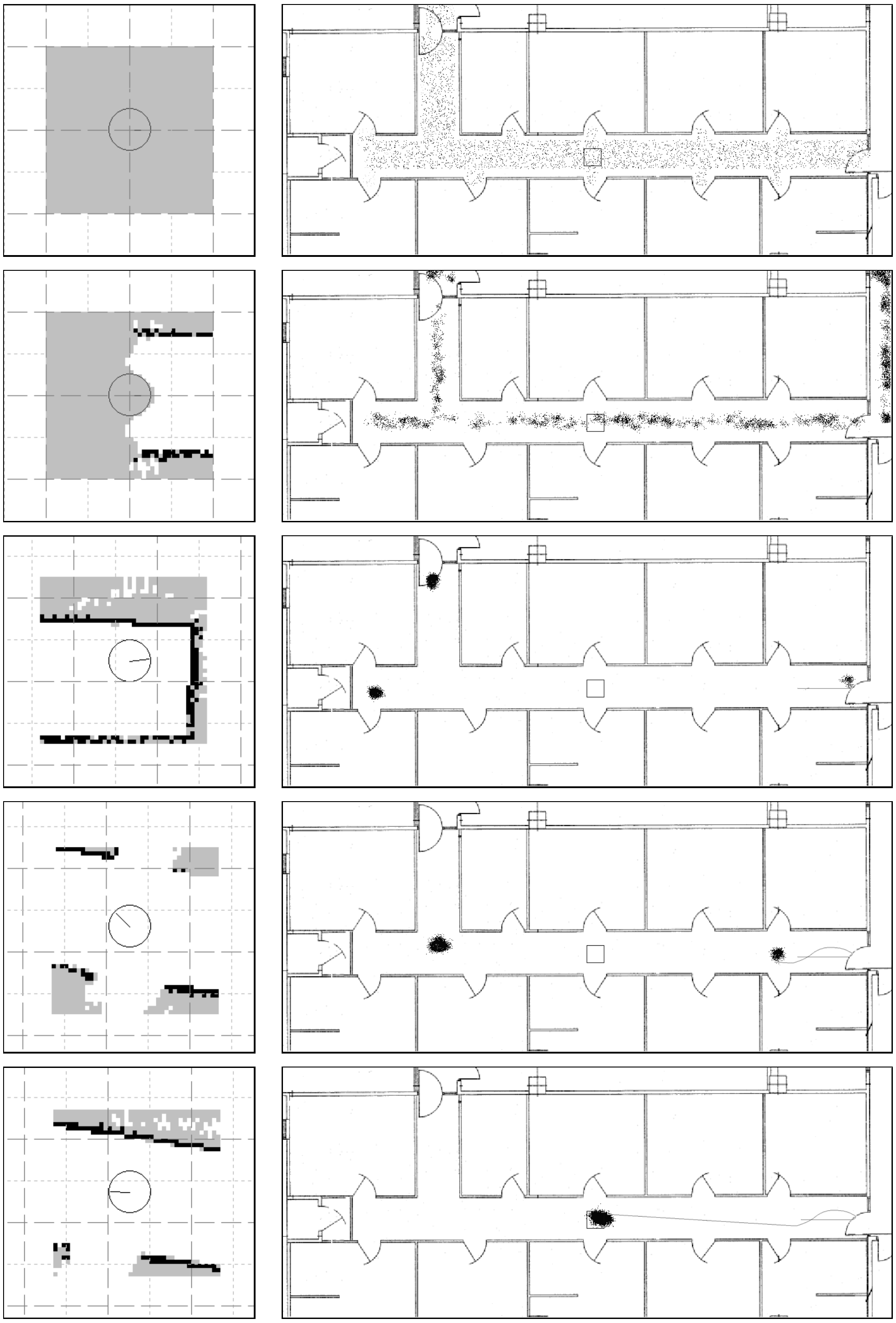


Figure 9.14: Typical trial in the real environment using the persistent strategy. The figures on the left show the robot's local occupancy map. The figures on the right show the evolution of the probability distribution $p(r)$. The dotted line shows the actual path taken by the robot; the goal is indicated by the square.

robot. The goal in all cases was the same. Some care must be taken when interpreting these results, in view of the relatively small number of trials (performing trials with a real robot, as opposed to a simulator, is quite time-consuming). Nevertheless, the performance of the persistent strategy appears to be quite good — the robot experienced only 1 failure in 12 trials, and that failure was due to the failure of the localisation system. Contrast this with the performance of the most probable and proportional voting strategies, which each failed on 4 occasions. These results confirm those obtained using the simulator.

Interestingly, the observed localisation failures were ultimately found to be a consequence of inaccuracies in the model, which was constructed manually from the architect's floor plans. Unfortunately, it was later discovered that the builder had taken a somewhat liberal interpretation of these plans, with some walls being misplaced by as much as one meter. This certainly highlights the need for robots that can construct their own model of the environment – a topic that is, unfortunately, beyond the scope of this thesis.

9.8 Conclusion

The probabilistic navigation system described in this chapter works, and works well. As the experiments show, the robot is able to determine its location, and reach its assigned goal, even when its initial location is *completely unknown*. There are, however, a couple of weaknesses with this approach that should be highlighted.

Firstly, models must be prepared by a human operator, and supplied, in their entirety, to the robot. As we saw in Section 9.7, however, models prepared in this way are prone to error – even if one has access to an appropriate set of floor-

	Most Probable	Prop. Voting	Persistent
Total trials	12	12	12
Failures	4	4	1
Mean time per trial (sec)	86 [±27]	123 [±60]	117 [±52]
Total failure rate (%)	33 [10, 72]	33 [10, 72]	8 [0, 49]
Localisation failure rate (%)	8	17	8
Strategy failure rate (%)	25	17	0

Table 9.3: Results for the real environment with unknown location.

plans, there is no guarantee (for a real environment) that these floor-plans will be accurate. Consequently, some mechanism by which the robot can *learn* a model of the environment, either autonomously (the ideal case) or under human supervision, would enhance the utility of this approach. A number of such mechanism have been described in recent literature on robot navigation [243, 246, 259, 260, 186]; I hope to address this topic in the near future.

The second weakness of this approach lies in the whole concept of a navigation strategy. While the persistent strategy is, in general, very effective, it is possible for the robot to find itself into a situation from which it cannot escape (one such situation was described in Section 9.6.3). While the experiments described in Section 9.7 do not contain any examples of this, these pathological situations have been observed to occur on a number of occasions (in keeping with Murphy's Law, this usually happens when the system is being demonstrated to a visiting dignitary). Consequently, there is much scope for further development of this aspect of the system.

Chapter 10

Conclusion

> "Would you tell me, please, which way I ought to walk from here?" [said Alice.]
>
> "That depends a great deal on where you want to get to," said the Cat.

from *Alice in Wonderland*, by Lewis Carroll.

10.1 Summary

This thesis describes an entire navigation system, all the way from the nuts-and-bolts of the actual robot hardware, through to some high-level probabilistic navigation software. The system is capable of navigating a robot to a distant goal, using only naturally occurring landmarks, even when the robot's initial location is completely unknown.

There are a number of elements of this system that work well, and other elements that would benefit from further refinement. There are, in addition, many possible enhancements and directions for further research. In this section, I will attempt to summarise some of these issues on chapter-by-chapter basis. I will also briefly consider some of the issues arising from the interaction between the local and global layers.

Introduction, Literature Review and Overview

Chapters 1, 2 and 3 present a general overview of the navigation system, and place this system in the context of the wider domain of mobile robotics research. This system is quite unique: very few real robotic systems are able to find their way to a distant goal when their initial location is unknown.

Implementation

Chapter 4 describes the hardware and software used to implement the navigation system. Overall, there is an emphasis on using commercially available off-the-shelf technology as a means of keeping costs down. The robot Tigger, for example, which costs less than $1000 to build, has been used for applications that would normally require much larger and more expensive robots. The developments in consumer technology that have made Tigger possible are expected to continue; consequently, both hardware and software will continue to evolve.

Building Local Maps Using Sonar

Chapter 5 describes a method for building local maps using ultrasonic range data. It introduces the concept of a *response grid*, which is intended to improve the robot's ability to build maps in highly specular environments. While the results obtained using this method are quite good, the map building process is very slow. In part, this is due to the limitations imposed by the rotating sonar sensor used on robot J Edgar. If sonar is to be used to support navigation at reasonable speeds (i.e. more than a few centimetres per second) this rotating sonar sensor must be replaced with a sonar ring.

Building Local Maps Using Vision

As an alternative to sonar, Chapter 6 describes a method for building local occupancy grids using images obtained from a single black-and-white camera. This method is certainly much faster than the sonar-based method described in Chapter 5: frame rates of over 10Hz are easily achievable. The approach does, however, have one limitation: it suffers from a limited field of view. Fortunately, this limitation can be addressed quite easily, either by mounting the camera on a pan head and swinging it from side to side (the current solution), or by making use of wide angle lenses and conical mirrors (the probable future solution).

The vision-based mapping method described in Chapter 6 also requires that the environment satisfies certain assumptions. There are, however, a number of possible extensions that would greatly enlarge the range of environments to which this simple method can be applied. These include the use of colour and texture information to help distinguish between obstacles and open space.

Local Navigation

Chapter 7 describes a method for local navigation that makes use of the local occupancy maps produced by the sonar and visual mapping modules. This chapter introduces the *dynamic utility fusion* approach (DUF), which is designed to support navigation at high speeds by taking into account the *dynamics* of the robot, as well as its kinematics. This idea has worked surprisingly well, and is one of the key elements in the overall success of the system. Irrespective of whatever else may be happening, the local navigation module will almost always ensure that the robot does not collide with obstacles. On those occasions when the robot *does* collide with an obstacle, it is invariably because the robot has failed to detect it.

Landmark Detection

Chapter 8 describes a method for detecting natural landmarks in the local occupancy grid using a simple template-based approach. These landmarks are used by the localisation module to aid in the determination of the robot's position. While the approach described in this chapter works well enough to support the localisation process, it does suffer from two problems.

The first problem is computational cost: even with early jump out, the template matching procedure can be computationally expensive. The landmark detection module does, in fact, require more processing power than any other module the system. This is one aspect of the overall system that definitely needs improving.

The second problem is that the detected landmarks are relatively sparse. Since the landmark templates are designed to match features such as walls and doorways, the robot may have to travel many meters between landmarks. This makes the localisation process much more difficult than it might otherwise be: it means that there is plenty of time for odometric errors to accumulate. One could address this problem by making use of other types of landmarks, whose distribution is much more dense. Unfortunately, as show in Chapter 8, there are drawbacks associated with all of the likely candidates. The search for landmarks that can be reliably detected and are densely distributed remains a subject of ongoing research.

Global Navigation

Chapter 9 describes a *probabilistic* approach to global navigation. This approach works extremely well: probabilistic localisation is very robust, even in the presence of gross errors in the global map; while navigation strategies provide a simple but effective solution to the problem of navigation under uncertainty. This chapter present what I believe are the two key contributions of the thesis:

- The *population-based* representation for probability distributions, which has a number of advantages over existing representations;

- The *persistent strategy*, which is capable of guiding the robot to a distant irrespective of the uncertainties in the robot's initial location.

Note, however, that Chapter 9 describes some situations in which the persistent strategy will fail; these situations remain the subject of ongoing research.

Local/Global Interaction

In principle, the system described in this thesis should be able to navigate the robot to the goal at up to 50 cm/sec (this is the maximum speed supported by the local navigation module). In practice, speeds must be kept much lower – less than 20 cm/sec. There are two reasons for this.

Firstly, unlike the local navigation module, the global planning module does not take into account the robot's dynamics. For example, if the robot is approaching an intersection at which the robot must make a turn, the global layer will not send the turn command to the local layer until the robot has actually entered the intersection. If the robot is travelling at high speed, it will be unable to make the turn, and will overshoot the turn-off. While the robot will subsequently turn around and attempt the turn once again, this behaviour is less than optimal. Clearly, if the robot is to travel at high speeds, the global planner must either anticipate such effects, or else the local navigation module must be extended somehow so that it has a greater understanding of the robot's *long term* goals. To some extent, this undermines the original reason for separating navigation in local and global components, which was that the local navigation module should not have to worry about long term problems, while the global navigation module should not have to worry about low-level obstacle avoidance problems. Hopefully, however, there is some middle ground that permits improved handling of robot dynamics, whilst still retaining this separation.

The second problem associated with high speeds is one of sensing: if the robot moves too fast, it will miss significant landmarks. For example, if the robot is moving at high speed down a corridor, there is a tendency to miss landmarks on either side, such as doorways. The robot may even sail straight through an intersection without detecting it. These problems highlight a issue that has not been addressed in this thesis: the problem of *directed* perception. In this thesis, perception as been treated as a *passive* process, in which the robot makes use of whatever data 'falls onto' its sensors. However, perception can also be treated as an *active* process, in which the robot actively seeks out relevant information in the world [78, 54]. One could, for example, extend the landmark detection module so that it looks for *partial* matches. Upon detection of such a partial match, the global layer could do one of two things: it could stop and inspect the possible landmark (if its position was highly uncertain); or it could ignore the possible landmark and continue (if its position was well known). The concept of active perception, at least in the context of computer vision, is a well researched topic [4]; many of these ideas may well be applicable to the problem of mobile robot navigation.

Both of the problems identified above arise as a result the somewhat limited way in which the local and global layers interact. The local layer sends pre-processed sensor data (in the form of an occupancy map) to the global layer, while the global layer sends back a simple goal command. To address the problems identified above, the architecture used in this system would have to be modified. One possible way to do this would be to extend the idea of *utility fusion*, used within the local navigation module, to cover the system as a whole (the resultant architecture would look not unlike Rosenblatt's DAMN architecture [204]). Thus, instead of simply sending the local navigation module a goal command, the global planning module would be able to comment on the utility of various sequences of actions. In doing so, it would be able take into account the robot's long-term goals. This would, for example, address the problem of overshooting turn-offs: those sequences that resulted in an overshoot would be assigned a low utility.

The same mechanism could be used to manage the robot's sensory activities: if the position of the robot was highly uncertain, the planning module would be able to assign a high utility to sequences that involved stopping and inspecting a potential landmark. It remains to be seen, however, whether the stochastic sequence selection mechanism used by the local navigation module would be effective when confronted with such complex requirements.

10.2 Closing Comments

In Chapter 1, it was suggested that many of the difficulties associated with mobile robot navigation derive from the impoverished perceptual abilities of mobile robots. Three different ways of addressing this problem were identified:

- Improve the robot's perceptual abilities to the point where they are commensurate with human abilities. The robot can then make use of the same naturally occurring landmarks that humans use.
- Modify the environment, through the addition of artificial beacons, to make it easier for the robot to find landmarks.
- Accept that landmarks may be difficult to find and inherently ambiguous, and instead improve the robot's localisation and planning abilities to the point where the robot can handle a great deal of uncertainty in its location.

While the first of these options represents the ideal case, there is no indication that there will be rapid progress on this front in the immediate future. The second option, which is both well researched and well understood, it not appropriate for all applications. This leaves the third option, which is the approach taken in this thesis. By making use of a probabilistic localisation technique, and coupling it to an appropriate navigation strategy, the system is able to handle a great deal of uncertainty in the robot's location. The experimental results presented in Chapter 9 indicate that this is indeed a viable approach.

Bibliography

[1] Y. Abe, T. Fukuda, F. Arai, Y. Yokoyama, and Y. Tanaka. Vision based navigation system considering error recovery for autonomous mobule robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 1993–1998, 1997.

[2] Y. Abe, M. Shikano, T. Fukuda, F. Arai, and Y. Tanaka. Vision based navigation system by variable template matching for autonomous mobile robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 952–957, 1998.

[3] H. Akbarally and L. Kleeman. A sonar sensor for accurate 3d target localisation and classification. In *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, volume 3, pages 3003–3008, 1995.

[4] J. Aloimonos, I. Weiss, and A. Bandyopadhyay. Active vision. *International Journal of Computer Vision*, 1(4):333–356, 1987.

[5] J. C. Alvarez, A. Shkel, and V. Lumelsky. Accounting for mobile robot dynamics in sensor-based motion planning: Experimental results. In *Proceedings*

of the 1998 IEEE International Conference on Robotics and Automation, volume 3, pages 2205–2210, 1998.

[6] O. Amidi, Y. Mesaki, and T. Kanade. Research on an autonomous vision-guided helicopter. In *Conference on Intelligent Robotics in Field, Factory, Service, and Space*, volume 2, pages 456–463, 1994.

[7] D. Apostolopoulos and J. Bares. Locomotion configuration of a robust rappelling robot. In *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 280–284, 1995.

[8] R. C. Arkin. Motor schema based navigation for a mobile robot: an approach to programming by behaviour. In *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, volume 1, pages 264–271, 1987.

[9] R. C. Arkin. Motor schema based mobile robot navigation. *International Journal of Robotics Research*, 8(4):92–112, 1989.

[10] J. M. Armingol, , L. Moreno, A. de la Escalera, and M. A. Salichs. Landmark perception planning for mobile robot localization. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 4, pages 3425–3430, 1998.

[11] M. Asada, Y. Fukui, and S. Tsuji. Representing global world of a mobile robot with relational local maps. *IEEE Transactions on Systems, Man and Cybernetics*, 20:1456–1460, 1990.

[12] S. Atiya and G. D. Hager. Real-time vision-based robot localization. *IEEE Transactions on Robotics and Automation*, 9(6):785–800, 1993.

[13] D. J. Austin and B. J. McCarragher. Discrete event control for mobile robots. In *Proceedings of the International Conference on Field and Service Robotics*, pages 364–369, 1997.

[14] N. Ayache and F. Lustman. Trinocular stereo vision for robotics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(1):73–84, 1991.

[15] T. Balch and R. Arkin. Avoiding the past: A simple but effective strategy for reactive navigation. In *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, volume 1, pages 678–685, 1993.

[16] J. Baldwin. Support logic programming. In A. J. et al, editor, *Fuzzy sets - Theory and Applications, Proceedings of NATO Advanced Study Institute*. Reidel Pub. Co., Norwell, MA, 1986.

[17] D. H. Ballard and C. M. Brown. *Computer Vision*. Prentice-Hall, 1982.

[18] D. Bapna, E. Rollins, A. Foessel, and R. Whittaker. Antenna pointing for high bandwidth communications from mobile robots. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, volume 4, pages 3468–3473, 1998.

[19] D. Bapna, E. Rollins, J. Murphy, E. Maimone, W. Whittaker, and D. Wettergreen. The atacama desert trek: Outcomes. In *pra98*, volume 1, pages 597–604, 1998.

[20] J. Bares and D. Wettergreen. Lessons learned from the development and deployment of dante ii. In *Proceedings of the International Conference on Field and Service Robotics*, pages 72–79, 1997.

[21] J. E. Bares and W. L. Whittaker. Walking robot with a circulating gait. In *Proceedings of the 1990 IEEE International Workshop on Intelligent Robots and Systems*, volume 2, pages 809–816, 1990.

[22] N. M. Barnes and Z. Q. Liu. Vision guided circumnavigating autonomous robots. In *ICSC '95 - Proceedings of the Third International Computer Science Conference*, pages 33–42, Dec. 1995.

[23] J. Battle, E. Mouaddib, and J. Salvi. Recent progress in coded structured light as a technique to solve the correspondance problem: a survey. *Pattern Recognition*, 31(7):963–982, 1998.

[24] M. Beckerman and E. M. Oblow. Treatment of systematic errors in the processing of wide angle sonar sensor data for robot navigation. *IEEE Transactions on Robotics and Automation*, 6(2):137–145, 1990.

[25] G. A. Bekey. *Autonomous Robots*, Special Issue on Autonomous Underwater Vehicles, volume 3, number 2/3, 1996.

[26] R. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, New Jersey, 1957.

[27] H. R. Beom and H. S. Cho. A sensor-based navigation for mobile robot using fuzzy logic and reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(3):464–477, 1995.

[28] J. O. Berger. *Statistical Decision Theory and Bayesian Analysis, Second Edition.* Springer-Verlag, New-York, 1985.

[29] T. O. Binford. Survey of model-based image analysis systems. *International Journal of Robotics Research*, 1(1):18–64, 1982.

[30] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man and Cybernetics*, 19(5):1179–1187, 1989.

[31] J. Borenstein and Y. Koren. The vector field histogram – fast obsstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, 1991.

[32] S. M. Bozic. *Digital and Kalman Filtering*. Edward Arnold (Hodder Headline Group), second edition, 1994.

[33] Ö. Bozma and R. Kuc. Building a sonar map in a specular environment using a single mobile sensor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(12), 1991.

[34] Ö. Bozma and R. Kuc. Novel sonar map-building based on physical principles of reflection. In *Proceedings of the 1991 IEEE International Conference on Systems, Man and Cybernetics*, volume 1, pages 75–80, 1991.

[35] Ö. Bozma and R. Kuc. Single sensor sonar map-building based on physical principles of reflection. In *IEEE/RSJ International Workshop on Intelligent Robots and Systems*, pages 1038–1043, 1991.

[36] D. J. Braunegg. Marvel: A system that recognises world locations from stereo vision. *IEEE Transactions on Robotics and Automation*, 9(3):303–308, 1993.

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

[38] R. A. Brooks. Visual map making for a mobile robot. In *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, pages 824–829, 1985.

[39] R. A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6:3–15, 1990.

[40] R. A. Brooks. Intelligence without reason. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 569–595, 1991.

[41] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47(1–3):139–159, 1991.

[42] R. A. Brooks, A. M. Flynn, and T. Marill. Self-calibration of motion and stereo vision for mobile robot navigation. In *Proceedings of the DARPA Workshop on Image Understanding*, pages 398–410. Morgan Kaufman, 1988.

[43] M. K. Brown. Feature extraction techniques for recognising solid objects with an ultrasonic range sensor. *IEEE Journal of Robotics and Automation*, RA-1(4), 1985.

[44] W. Burgard, A. B. Cremers, D. Fox, D. Haehnel, G. Lakemeyer, D. Shultz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. Technical Report CMU-CS-98-139, Carnegie Mellon University, 1998.

[45] W. Burger and B. Bhanu. Estimating 3-d egomotion from perspective image sequences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(11):1040–1058, 1990.

[46] G. Campion, G. Bastin, and B. D'Andréa-Novel. Structural properties and classification of kinematic and dynamic models of wheeled mobile robots. *IEEE Transactions on Robotics and Automation*, 12(1):47–62, 1996.

[47] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, Nov. 1986.

[48] Z. Cao and E. L. Hall. Beacon recognition in omni-vision guidance. *Proceedings-of-the-SPIE –The-International-Society-for-Optical-Engineering*, 1230:788–790, 1990.

[49] C. C. Chang and K.-T. Song. Dynamic motion planning based on real-time obstacle prediction. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, volume 3, pages 2402–2407, 1996.

[50] C. C. Chang and K.-T. Song. Environment prediction for a mobile robot in a dynamic environment. *IEEE Transactions on Robotics and Automation*, 13(6):862–872, 1997.

[51] G. Cheng and A. Zelinsky. Real-time visual behaviours for navigating a mobile robot. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*, volume 2, pages 973–980, 1996.

[52] S. Clark and H. Durrant-Whyte. The design of a high performance mmw radar system for autonomous land vehicle navigation. In *Proceedings of the International Conference on Field and Service Robotics*, pages 292–299, 1997.

[53] S. Clark and H. Durrant-Whyte. Autonomous land vehicle navigation using mmw radar. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, volume 1, pages 16–20, 1998.

[54] D. J. Cook, P. Gymtrasiewicz, and L. B. Holder. Decision-theoretic cooperative sensor planning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(10):1013–1023, 1996.

[55] D. Cooms, M. Herman, T.-H. Hong, and M. Nashman. Real-time obstacle avoidance using central flow divergence, and peripheral flow. *IEEE Transactins on Robotics and Automation*, 14(1):49–59, 1998.

[56] J. Cooper, S. Venkatesh, and L. Kitchen. Early jump out corner detectors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(8):823–828, 1993.

[57] P. Corke, J. Roberts, and G. Winstanley. 3d perception form mining robots. In *Proceedings of the International Conference on Field and Service Robotics*, pages 41–47, 1997.

[58] P. Corke, G. Winstanley, and M. Roberts. Modelling and control of a 3500 tonne mining robot. In *Experimental Robotics V. The Fifth International Symposium*, pages 262–274, 1998.

[59] J. D. Courtney and A. K. Jain. Mobile robot localization via classification of multisensor maps. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 1672–1678, 1994.

[60] I. J. Cox. Blanch — an experiment in guidance and navigation of an autonomous robot vehicle. *IEEE Transactions on Robotics and Automation*, 7(2):193–204, April 1991.

[61] J. D. Crisman and C. E. Thorpe. Scarf: A color vision system that tracks roads and intersections. *IEEE Transactions on Robotics and Automation*, 9(1):49–58, 1993.

[62] J. L. Crowley, P. Bobet, K. Sarachick, S. Mely, and M. Kurek. Mobile robot perception using vertical line stereo. *Robotics and Autonomous Systems*, 7(2–3):125–138, 1991.

[63] E. R. Davies. *Machine Vision: Theory, Algorithms, Practicalities.* Academic Press, Harcourt Brace Jovanovich, 1990.

[64] U. R. Dhond and J. K. Aggarwal. Structure from stereo – a review. *IEEE Transactions on Systems, Man and Cybernetics*, 19(6):1489–1510, 1989.

[65] W. C. Dickson and R. H. Cannon, Jr. Experimental results of two free-flying robots capturing and manipluating a free-flying object. In *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 51–58, 1995.

[66] G. Dissanayake, M. Herbert, A. Stentz, and H. Durrant-Whyte. Map building and terrain-aided localisation in an underground mine. In *Proceedings of the International Conference on Field and Service Robotics*, pages 34–40, 1997.

[67] G. Dudek, P. Freeman, and S. Hadjres. Using local information in a non-local way for mapping graph-like worlds. In *Proceedings of the Thirteenth Interna-*

tional Joint Conference on Artificial Intelligence, volume 2, pages 1639–1645, 1993.

[68] G. Dudek, P. Freeman, and S. Hadjres. Mapping in unknown graph-like worlds. *Journal of Robotics Systems*, 13(8):539–559, 1996.

[69] G. Dudek, K. Romanik, and S. Whitesides. Localizing a robot with minimum travel. In *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms*, pages 437–446, 1995.

[70] H. S. Dulimarta and A. K. Jain. Mobile robot localisation in indoor environment. In *Proceedings of the Third International Conference on Automation, Robotics and Computer Vision*, volume 3, pages 2204–2208, 1994.

[71] H. S. Dulimarta and A. K. Jain. Mobile robot localisation in indoor environments. *Pattern Recognition*, 30(1):99–111, 1997.

[72] H. Durrant-Whyte. An automous guided vehicle for cargo handling applications. *International Journal of Robotics Research*, 15(5):407–440, October 1996.

[73] A. Elfes. A distributed control architecture for an autonomous mobile robot. *Artificial Intelligence*, 1(2), 1986.

[74] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, RA-3(3):249–265, 1987.

[75] A. Elfes. *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation.* PhD thesis, Electrical and Computer Engineering Department/Robotics Institute, Carnegie-Mellon University, May 1989.

[76] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6), 1989.

[77] A. Elfes. Occupancy grids: A stochastic spatial representation for active robot perception. In *Proceedings of the Sixth Conference on Uncertainty in AI*. Morgan Kaufmann Publishers, Inc, July 1990.

[78] A. Elfes. Dynamic control of robot perception using stochastic spatial models. In *Proceedings of the International Workshop on Information Processing in Mobile Robots*, pages ??–??, 1991.

[79] R. Ellepola and P. Kovesi. Mobile robot navigation in a semi-structured environment. In *Proceedings of the Fourth International Conference on Control, Automation and Robotics*, pages 914–918, 1996.

[80] R. Ellepola, P. Kovesi, and K. Wessen. Mobile robot navigation using passive beacons with active sensors. In *Conference Proceedings DICTA-95: Digital Image Computing, Technologies and Applications*, pages 679–685, 1995.

[81] A. Elnagar and A. Basu. Smooth and acceleration minimizing trajectories for mobile robots. In *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2215–2221, 1993.

[82] A. Elnagar and A. Basu. Piecewise smooth and safe trajectory planning. *Robotica*, 12(4):299–307, 1994.

[83] D. Estival. Integration of speech and vision in a small mobile robot. In *Proceedings of the Workshop on Content Visualization and Intermedia Representation (CVIR'98) Montreal*, pages 109–114, 1998.

[84] D. Estival. Adding language capabilities to a small robot. In *Proceedings of the Sixth International Symposium of Social Communication (ISSC) Santiago de Cuba*, January 1999. to appear.

[85] A. H. Fagg, M. A. Lewis, J. F. Montgomery, and G. A. Bekey. The usc autonomous flying vehicle: an experiment in real-time behavior-based control. In *Proceedings 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 1173–1180, 1993.

[86] R. M. Feldman. *Applied probability and stochastic processes.* Boston : PWS Pub. Company, 1996.

[87] Y. Feng and K. Kubik. Real time gps dynamic-positioning approaches to mobile robot navigation. In *Fourth International Symposium on Signal Processing and its Applications*, volume 1, pages 258–261, 1996.

[88] C. Fennema, A. Handson, E. Riseman, J. R. Beveridge, and R. Kumar. Model-directed mobile robot navigation. *IEEE Transactions on Systems, Man and Cybernetics*, 20(6):1352–1369, 1990.

[89] J. Firby. Task networks for controlling continuous processes. In *Proceedings of the Second International Conference on AI Planning Systems*, 1994.

[90] R. J. Firby, P. N. Prokopowicz, and M. J. Swain. The animate agent architecture. In *Artificial Intelligence and Mobile Robots*. AAAI Press/MIT Press, 1997.

[91] A. M. Flynn, R. A. Brooks, W. M. Wells III, and D. S. Barrett. Intelligence for miniature robots. *Sensors and Actuators*, 20(1–2):187–196, 1989.

[92] A. M. Flynn, R. A. Brooks, W. M. Wells III, and D. S. Barrett. The world's largest one cubic inch robot. In *Proceedings of the IEEE Conference on Micro Electro Mechanical Systems*, pages 98–101, 1989.

[93] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 4(1):23–33, 1997.

[94] D. Fox, W. Burgard, S. Thrun, and A. B. Cremers. A hybrid collision avoidance method for mobile robots. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, volume 2, pages 1238–1243, 1998.

[95] A. Fujimori, P. N. Nikiforuk, and M. M. Gupta. Adaptive navigation of mobile robots with obstacle avoidance. *IEEE Transactions on Robotics and Automation*, 13(4):596–602, 1997.

[96] T. Fukuda, Y. Yokoyama, F. Arai, K. Shimojima, S. Ito, Y. Abe, K. Tanaka, and Y. Tanaka. Navigation systems based on ceiling landmark recognition for autonomous mobile robot – position/orientation control by landnark recognition with plus and minus primitives. In *Proceedings IEEE International Conference on Robotics and Automation*, volume 2, pages 1720–1725, 1996.

[97] E. Gat. Three-layer architectures. In *Artificial Intelligence and Mobile Robots*. AAAI Press/MIT Press, 1997.

[98] E. Gat, R. Desai, R. Ivlev, J. Loch, and D. P. Miller. Behaviour control for robotic exploration of planetary surfaces. *IEEE Transactions on Robotics and Automation'*, 10(4), August 1994.

[99] S. Ghosh. *Analytical Photogrammetry*. Pergamon Press, 1979.

[100] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[101] H. Goldstein. *Classical Mechanics.* Addison-Wesley, 1980.

[102] R. Greiner and R. Isukapalli. Learning to select useful landmarks. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 26(3):437–449, 1996.

[103] E. Grosso and M. Tistarelli. Active/dynamic stereo vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(9):868–979, 1995.

[104] J. Guldner and V. I. Utkin. Sliding mode control for gradient tracking and robot navigation using artificial potential fields. *IEEE Transactions on Robotics and Automation*, 11(2):247–254, 1995.

[105] J. Guldner and V. I. Utkin. Tracking the gradient of artificial potential fields: sliding mode control for mobile robots. *International Journal of Control*, 63(3):417–432, 1996.

[106] G. D. Hager, D. Kriegman, E. Teh, and C. Rasmussen. Image-based prediction of landmark features for mobile robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 1040–1046, 1997.

[107] S. Haihang, G. Muhe, and H. Kerzhong. An intergrated gps/ceps position estimation system for outdoor mobile robot. In *Proceedings of the 1997 IEEE International Conference on Intelligent Processing Systems*, volume 2, pages 1282–1286, 1997.

[108] K. D. Harris and M. Recce. Neural model of a grid-based map for robot sonar. In *Proceedings of the IEEE Symposium on Computational Intelligence, Robotics and Automation*, pages 34–39. IEEE, 1997.

[109] M. Hashima, F. Hasegawa, S. Kanda, T. Maruyama, and T. Uchiyama. Localization and obstacle detection for robots carrying food trays. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 345–351, 1997.

[110] K. Hirai. Current and future perspective of honda humanoid robot. In *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 500–508, 1997.

[111] G. Hirzinger. Robots in space – a survey. *Advanced Robotics*, 9(6):625–651, 1995.

[112] R. Hoffman and E. Krotkov. Terrain mapping for outdoor robots: Robust perception for walking in the grass. In *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, volume 1, pages 529–533, 1993.

[113] A. A. Holenstein and E. Badreddin. Collision avoidance in a behaviour-based mobile robot design. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, volume 1, pages 898–903, 1998.

[114] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* MIT Press, 1992. Original edition published 1975.

[115] J. Hollingum. Unmanned vehicles go to war. *Industrial Robot*, 25(6):379–383, 1998.

[116] J. A. Horst. Maintaining multi-level planar maps in intelligent systems. In *Proceedings of the IEE International Conference on Robotics and Automation*, volume 3, pages 1061–1066, 1996.

[117] I. Horswill. Polly: a vision-based artificial agent. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 824–829, 1993.

[118] I. Horswill. Visual collision avoidance by segmentation. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*, volume 2, pages 902–909, 1994.

[119] A. Howard and L. Kitchen. Generating sonar maps in highly specular environments. In *Proceedings of the Fourth International Conference on Control, Automation, Robotics, and Vision*, pages 1870–1874, 1996.

[120] A. Howard and L. Kitchen. Fast visual mapping for mobile robot navigation. In *Proceedings of the IEEE International Conference on Intelligent Processing Systems*, pages 1251–1255, 1997.

[121] A. Howard and L. Kitchen. Navigation using natural landmarks. *Journal of Robotics and Automous Systems (Special Issue on Field and Service Robotics)*, 1998. To Appear.

[122] A. Howard and L. Kitchen. Vision-based navigation using natural landmarks. In A. Zelinsky, editor, *Field And Service Robotics*. Springer-Verlag Engineering Series, 1998.

[123] K. Inagaki. Gait study for a hexapod walking with disabled leg. In *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 408–413, 1997.

[124] Innomedia. Infowave wireless computer connection. http://www.innomedia.com/products/infowave1.html, 1999.

[125] H. Inoue, M. Inaba, T. Mori, and T. Tachikawa. Real-time robot vision system based on correlation technology. In *Proceedings of the International Symposium on Industrial Robots (ISIR)*, pages 675–680, 1993.

[126] R. A. Jarvis. A perspective on range finding techniques for computer vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(2), 1983.

[127] R. A. Jarvis. On distance transform based collision free path planning for robot navigation in known, unknown and time-varying environments. In Y. F. Zang, editor, *Advanced Mobile Robots*, pages 3–31. World Scientific, 1994.

[128] R. A. Jarvis and A. Lipton. Go-2:- an autonomous mobile robot for a science museum. In *Proceedings of the Fourth International Conference on Control, Automation, Robotics, and Vision*, pages 260–266, 1996.

[129] K-Team. Khepera mobile robots. http://www.k-team.com/, 1999.

[130] L. P. Kaelbling, A. R. Cassandra, and J. A. Kurien. Acting under uncertainty: Discrete bayesian models for mobile roobot navigation. In *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 963–972, 1996.

[131] N. Kato and T. Inaba. Guidance and control of a fish robot with apparatus of pectoral fin motion. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, volume 1, pages 446–451, 1998.

[132] A. Kelly and A. T. Stentz. Analysis of requirements for high speed rough terrain mobility: Part 1 throughput and response. In *ICRA*, 1997.

[133] A. Kelly and A. T. Stentz. Analysis of requirements for high speed rough terrain mobility: Part 2 - resolution and accuracy. In *ICRA*, 1997.

[134] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.

[135] J. H. Kim and H. S. Cho. Real-time determination of a mobile robot's position by linear scanning of a landmark. *Robotica*, 10(4):309–319, 1992.

[136] L. Kleeman. Optimal estimation of position and heading for moble robots using ultrasonic beacons and dead-reckoning. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, volume 3, pages 2582–2587, 1992.

[137] L. Kleeman and R. Kuc. Mobile robot sonar for target localisation and classification. *International Journal of Robotics Research*, 14(4):295–318, 1995.

[138] S. Koenig and R. Simmons. Real-time search in nondeterministic domains. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1660 – 1667, 1995.

[139] S. Koenig and R. Simmons. Unsupervised learning of probabilistic models for robot navigation. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, volume 3, pages 2301–2308, 1996.

[140] S. Koenig and R. Simmons. Xavier: A robot navigation architecture based on partially observable markov decision process models. In D. Kortenkamp, R. Bonasso, and R. Murphy, editors, *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, chapter 4, pages 91–122. MIT Press, 1997.

[141] K. C. Koh, J. S. Kim, and H. S. Cho. A position estimation system for mobile robots using monocular image of a 3-d landmark. *Robotica*, 12(5):431–441, 1994.

[142] J. Kohlas. *A mathematical theory of hints : an approach to the Dempster-Shafer theory of evidence*. Springer-Verlag, 1995.

[143] T. Komatsu, M. Uenohara, S. Iikura, H. Miura, and I. Shimoyama. The development of an autonomous space robot operation testbed: Asrot. *Advanced Robotics*, 8(1):45–59, 1994.

[144] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *IEEE International Conference on Robotics and Automation*, pages 1398–1404, 1991.

[145] D. Kortenkamp, M. Huber, C. Cohen, U. Raschke, F. Koss, and C. Congdon. Intergrating high-speed obstacle avoidance, global path planning and vision sensing on a mobile robot. In D. Kortenkamp, R. P. Bonasso, and R. Murphy,

editors, *Artificial Intelligence and Mobile Robots : Case Studies of Successful Robot Systems*, pages 53–71. MIT Press, 1998.

[146] D. Kortenkamp, M. MacMahon, D. Ryan, R. P. Bonasso, and L. Moreland. Applying a layered control architecture to a free-flying space camera. In *Proceedings of the IEEE International Joint Symposia on Intelligence and Systems*, pages 188–194, 1998.

[147] S. Kotani, K. Kaneko, T. Shinoda, and H. Mori. Mobile robot navigation based on vision and dgps information. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, volume 3, pages 2524–2529, 1998.

[148] D. J. Kriegman, E. Triendl, and T. O. Binford. Stereo vision and navigation in buildings for mobile robots. *IEEE Transactions on Robotics And Automation*, 5(6):792–803, 1989.

[149] B. H. Krogh and C. E. Thorpe. Integrated path planning and dynamic vehicle steering control for automated vehicles. In *Proceedings of the 1986 IEEE International Conference on Robotics and Automation*, pages 1664–1669, 1986.

[150] R. Kuc and V. B. Viard. A physically based navigation strategy for sonar-guided vehicles. *International Journal of Robotics Research*, 10(2):75–87, 1991.

[151] B. Kuipers and Y.-T. Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Journal of Robotics and Autonomous Systems*, 8:47–63, 1991.

[152] A. Kurz. Constructing maps for mobile robot navigation based on ultrasonic range data. *IEEE Transactions on Systems, Man and Cybernetics. Part B: Cybernetics*, 26(2):233–242, 1996.

[153] K. J. Kyriakopoulos and G. N. Saridis. Optimal and suboptimal motion planning for collision avoidance of mobile robots in non-stationary environment. *Journal of Intelligent and Robotics Systems*, 11:223–267, 1995.

[154] X. Lebègue and J. K. Aggarwal. Significant line segments for an indoor mobile robot. *IEEE Transactions on Robotics and Automation*, 9(6):801–815, 1993.

[155] D. Lee and M. Recce. Quantitative evaluation of the exploration strategies of a mobile robot. *International Journal of Robotics Research*, 16(4):413–417, 1997.

[156] E. Lenko and T. Varsavsky. Fuzzy robot interaction. Fourth year Computer Engineering project: final report, 1998.

[157] J. J. Leonard and H. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems*, volume 3, pages 1442–1447, 1991.

[158] J. J. Leonard and H. Durrant-Whyte. *Directed Sonar Sensing for Mobile Robot Navigation*. Kluwer, 1992.

[159] L. M. Lorigo, R. A. Brooks, and E. L. Grimson. Visually-guided obstacle avoidance in unstructured environments. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*, volume 1, pages 373–379, 1997.

[160] J. D. Lowrance, T. M. Strat, L. P. Wesley, T. D. Garvey, E. H. Ruspini, and D. E. Wilkins. The theory, implementation, and practice of evidential reasoning. Technical report, SRI Project 5071, 1991. Final report.

[161] T. Lozano-Perez and M. Mason. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1):3–24, 1984.

[162] S. Maeda, Y. Kuno, and Y. Shirai. Active navigation vision based on eigenspace analysis. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1018–1023, 1997.

[163] M. J. Mataric. Integration of representation into goal-driven behaviour-based robots. *IEEE Transactions on Robotics and Automation*, 8(3):304–312, 1992.

[164] Y. Matsumoto, M. Inaba, and H. Inoue. Visual navigation using view-sequenced route representation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 586–591, 1992.

[165] Y. Matsumoto, M. Inaba, and H. Inoue. Memory-based navigation using omni-view sequence. In *Proceedings of the International Conference on Field and Service Robotics*, pages 184–191, 1997.

[166] C. McCarthy. A web based autonomous mobile robot: Project proposal. Honours Thesis Proposal – Department of Computer Science, University of Melbourne, May 1998.

[167] D. McFalls and E. Franke. A robotic assistent for space station freedom. *Robotics Today*, 2(2):1–6, 1989.

[168] A. L. Meyrowitz, D. R. Blidberg, and R. C. Michelson. Autonomous vehicles. *Proceedings of the IEEE*, 84(8):1147–1164, 1996.

[169] T. Mochida, A. Ishiguro, T. Aoki, and Y. Uchikawa. Behaviour arbitration for autonomous mobile robots using emotion mechanisms. In *Proceedings of the*

1995 IEEE?RSJ International Conference on Intelligent Robots and Systems, volume 3, pages 516–521, 1995.

[170] H. Moravec. *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover.* PhD thesis, Stanford University, Stanford, California, 1980. Available as CMU-RI-TR-3.

[171] H. Moravec. Rover visual obstacle avoidance. In *Proceedings of the seventh International Joint Conference on Artificial Intelligence*, pages 785–790, 1981.

[172] H. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, pages 61–74, Summer 1988.

[173] R. Mukherjee and Y. Nakamura. Formulation and efficient computation of inverse dynamics of space robots. *IEEE Transactions on Robotics And Automation*, 8(3):400–406, 1992.

[174] V. S. Nalwa and T. O. Binford. On detecting edges. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):699–714, 1986.

[175] E. Nebot and H. Durrant-Whyte. Using quaternions in a low cost strapdown ins unit. In *Proceedings of the International Conference on Field and Service Robotics*, pages 286–291, 1997.

[176] S. Negahdaripour, B. Y. Hayashi, and Y. Aloimonous. Direct motion stereo for passive navigation. *IEEE Transactions on Robotics and Automation*, 11(6):829–843, 1995.

[177] W. Neubauer. A spider-like robot that climbs verticall in ducts or pipes. In *Proceedings of the 1994 IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, volume 2, pages 1178–1185, 1994.

[178] M. Nilsson. Snake robot free climbing. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, volume 4, pages 3415–3420, 1997.

[179] N. J. Nilsson. *Principles of Artificial Intelligence.* Tioga Pub Co, Palo Alto, California, 1980.

[180] H. K. Nishihara. Practical real-time imaging stereo matcher. *Optical Engineering*, 23(5):536–545, 1984.

[181] H. K. Nishihara and T. Poggio. Stereo vision for robotics. In *Robotics Research, First International Symposium*, pages 489–505. MIT Press, 1984.

[182] I. Nourbakhsh and M. Genesereth. Assumptive planning and execution: a simple, working robot architecture. *Autonomous Robots Journal*, 3(1):49–67, 1996.

[183] A. Ohya, Y. Nagashima, and S. Yuta. Exploring unknown environment and map construction using ultrasonic sensing of normal direction of walls. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, pages 485–492, 1994.

[184] A. Okabe, B. Boots, and K. Sugihara. *Spatial tessellations : concepts and applications of Voronoi diagrams.* Wiley, 1992.

[185] M. Ollis and A. Stentz. Vision-based perception for an automated harvester. In *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 1838–1844, 1997.

[186] G. Oriolo, G. Ulivi, and M. Vendittelli. Real-time map building and navigation for autonomous robots in unknown environments. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, 28(3):316–333, 1998.

[187] A. Oyha, E. Shoji, and S. Yuta. 3-d range sensor using fiber grating for recognition of autonomous mobile robot's passage space. In *IROS '94.Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, volume 3, pages 1759–1763, 1994.

[188] O. Ozeki, T. Nakano, and S. Yamamoto. Real-time range measurement device for three-dimensional object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(4):550–554, 1986.

[189] D. Pagac, E. M. Nebot, and H. D. Whyte. An evidential approach to probabilistic map-building. In *Proceedings of the 1996 IEEE International Conference of Robotics and Automation*, volume 1, pages 745–750, 1996.

[190] D. Pagac, E. M. Nebot, and H. D. Whyte. An evidential approach to map-building for autonomous vehicles. *IEEE Transactions on Robotics and Automation*, 14(4):623–629, 1998.

[191] J. B. Paris. *The uncertain reasoner's companion : a mathematical perspective.* Cambridge University Press, 1994.

[192] D. W. Payton, J. K. Rosenblatt, and D. M. Keirsey. Plan guided reaction. *IEEE Transactions on Systems, Man and Cybernetics*, 20(6):1370–1382, 1990.

[193] N. Pears. Optical triangulation range sensors for vehcile manoevres. In S. Cameron and P. Probert, editors, *Advanced Guided Vehicles: Aspects of the Oxford AGV Project*, pages 85–106. World Scientific Series in Robotics and Automated Systems, World Scientific, 1994.

[194] R. R. Playter and M. H. Raibert. Control of a biped somersault in 3d. In *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1992.

[195] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C : the Art of Scientific Computing.* Cambridge University Press, 1988.

[196] A. Price, A. Jennings, and J. Kneen. Omnidirectional robots: New challenges in mobility, vision and tactics for robot soccer. In *Proceedings of the International Conference on Field and Service Robotics*, pages 493–498, 1997.

[197] M. H. Raibert, H. B. Brown Jr, and M. Cheppois. Experiments in balance with a 3d one-legged hopping machine. *International Journal of Robotics Research*, 3:75–92, 1984.

[198] M. H. Raibert, M. Chepponis, and H. B. Brown Jr. Running on four legs as though they were one. *IEEE Journal of Robotics and Automation*, 2:70–82, 1986.

[199] P. G. Research. Triclops vision system. http://www.ptgrey.com, 1998.

[200] E. Rich and K. Knight. *Artificial Intelligence.* McGraw-Hill, 1991.

[201] I. Robotics. Ariel: Autonomous legged underwater vehicle. http://www.isrobotics.com/research/ariel.html, 1999.

[202] I. Robotics. Darts – devices for rapid acceleration and turning. http://www.isrobotics.com/research/darts.html, 1999.

[203] J. K. Rosenblatt. Damn: a distributed architecture for mobile navigation. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2–3):339–360, 1997.

[204] J. K. Rosenblatt. Utility fusion: Map-based planning in a behaviour-based system. In *Proceeding of the International Conference on Field and Service Robotics*, pages 426–433, 1997.

[205] J. K. Rosenblatt and C. E. Thorpe. Combining multiple goals in a behaviour-based architecture. In *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 136–141, 1995.

[206] T. Rossmann and F. Pfeiffer. Control of an eight-legged pipe crawling robot. In *Experimental Robotics V. The Fifth International Symposium*, pages 335–346, 1998.

[207] V. Roveda. A parallel algorithm to find an match known objects to help the navigation of an autonomous mobile robot. In *Computer Intergrated Manufacturing, Proceedings of the Seventh CIM-Europe Annual Conference*, pages 373–381, 1991.

[208] A. Russell, D. Thiel, and A. Mackay. Sensing odour trails for mobile robot navigation. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, volume 3, pages 2672–2677, 1994.

[209] R. A. Russell and D. Thiel. Recruiting swarm robots using coded odour trails. In *Proceedings of the International Conference on Field and Service Robotics*, pages 472–476, 1997.

[210] A. Saffiotti. Blending reactivity and goal-directedness in a fuzzy controller. In *Proceedings of the 2nd Fuzzy-IEEE Conference*, pages 134–139, 1993.

[211] A. Saffiotti. The uses of fuzzy logic in autonomous robotics: a catalogue raisonne. *Soft Computing*, 1(4):180–197, 1997.

[212] P. Saint-Marc, J. L. Jezouin, and G. Medoni. A versatile pc-based range finding system. *IEEE Transactions on Robotics and Automation*, 7(2), 1991.

[213] J. Santos-Victor, G. Sandini, F. Curotto, and S. Garibaldi. Divergent stereo in autonomous navigation: From bees to robots. *International Journal of Computer Vision*, 14:159–177, 1995.

[214] S. Scheding, E. M. Nebot, M. Stevens, H. Durrant-Whyte, J. Roberts, P. Corke, J. Cunningham, and B. Cook. Experiments in autonomous underground guidance. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, volume 3, pages 1898–1903, 1997.

[215] B. Schiele and J. L. Crowley. A comparison of position estimation techniques using occupancy grids. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 1628–1634, 1994.

[216] B. Schiele and J. L. Crowley. A comparison of position estimation techniques using occupancy grids. *Robotics and Autonomous Systems*, 12(3–4):163–171, 1994.

[217] V. Sequeira, J. G. M. Goncalves, and M. I. Ribeiro. 3d environment modelling using laser range sensing. *Robotics and Autonomous Systems*, 16(1):81–91, 1995.

[218] G. Shafer. *A Mathematical Theory of Evidence.* Princeton University Press, 1976.

[219] S. Shafer, A. Stentz, and C. Thorpe. An architecture of sensor fusion in a mobile robot. In *Proceedings of the 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 2002–2010, 1986.

[220] Y. Shan and Y. Koren. Design ad motion planning of a mechanical snake. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(4):1091–1100, 1993.

[221] D. Sherman, G. Vidal, G. Marchica, and R. Johnson. University of central florida technical report: Auvs international aerial robotics competition. In *AUVSI '96 Proceedings. AUVSI - Assoc. Unmanned Vehicle Syst. Int*, pages 955–964, 1996.

[222] A. M. Shkel and V. J. Lumelsky. The jogger's problem: Accounting for body dynamics in real-time motion planning. In *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots ad Systems*, volume 2, pages 441–447, 1995.

[223] A. M. Shkel and V. J. Lumelsky. Incorporating body dynamics into sensor-based motion planning: The maximum turn strategy. *IEEE Transactions on Robotics and Automation*, 13(6):873–880, 1997.

[224] R. Simmons and S. Koenig. Probabilistic navigation in partially observable environments. In *Proceedings of International Joint Conference on Artificial Intelligence*, volume 2, pages 1080–1087, 1995.

[225] R. G. Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1), February 1994.

[226] K. T. Simsarian, T. J. Olsen, and N. Nandhakumar. View-invariant regions and mobile robot self-localization. *IEEE Transactions on Robotics and Automation*, 12(5):811–816, 1996.

[227] S. Singh and A. Kelly. Robot planning in the space of feasible actions: Two examples. In *Proc. Int'l Conference on Robotics and Automation*, April 1996.

[228] D. R. Smith and T. Kanade. Autonomous scene description with range imagery. *Computer Vision, Graphics and Image Processing*, 31(3):322–334, 1985.

[229] M. V. Srinivasan, J. S. Chahl, M. G. Nagle, K. Weber, S. Venkatesh, and S. W. Zhang. Robot navigation inspired by principles of insect vision. In *Proceedings of the International Conference on Field and Service Robotics (FRS'97)*, pages 18–20, 1997.

[230] M. V. Srinivasan, S. W. Zhang, M. Lehrer, and T. S. Collett. Honeybee navigation en route to the goal: visual flight control and odometry. *Journal of Experimental Biology*, 199:237–244, 1996.

[231] A. Stentz. The focussed d* algorithm for real-time replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, August 1995.

[232] S. Sukkarieh, E. Nebot, and H. Durrant-Whyte. The gps aiding of ins for land vehicle navigation. In *Proceedings of the International Conference on Field and Service Robotics*, pages 278–285, 1997.

[233] H. Takeda, C. Facchinetti, and J.-C. Latombe. Planning the motions of a mobile robot in a sensory uncertainty field. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(10):1002–1017, 1994.

[234] R. Talluri and J. K. Aggarwal. Position estimation for an autonomous mobile robot in an outdoor environment. *IEEE Transactions on Robotics and Automation*, 8(5):573–584, 1992.

[235] R. Talluri and J. K. Aggarwal. Image/map correspondence for mobile robot self-location using computer graphics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(6):597–601, 1993.

[236] R. Talluri and J. K. Aggarwal. Mobile robot self-location using model-image feature correspondence. *IEEE Transactions on Robotics and Automation*, 12(1), February 1996.

[237] J. Thomas, G. Wilson, M. Wakefield, A. Lampert, and P. Malkin. Integrated natural speech and automated navigation engine (insane). Third year Software Engineering project, 1998.

[238] C. Thorpe, M. Herbert, T. Kanade, and S. Shafer. Toward autonomous driving: the cmu navlab. *IEEE Expert*, 6(4):31–42, 1991.

[239] C. Thorpe, M. Herbert, T. Kanade, and S. Shafer. Toward autonomous driving: the cmu navlab. ii. architecture and systems. *IEEE Expert*, 6(4):42–52, 1991.

[240] C. Thorpe, M. H. Herbert, T. Kanade, and S. A. Shafer. Vision and navigation for the carnegie-mellon navlab. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(3):362–373, 1998.

[241] S. Thrun. Bayesian landmark learning for mobile robot localization. *Machine Learning*, 33(1):41–76, 1998.

[242] S. Thrun. Finding landmarks for mobile robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 958–963, 1998.

[243] S. Thrun. Learning metri-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[244] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schmidt. Map leaning and high-speed navigation in rhino. In D. Kortenkamp, R. P. Bonasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots : Case Studies of Successful Robot Systems*, pages 21–50. MIT Press, 1998.

[245] S. Thrun and A. Buecken. Integrating grid-based and topological maps for mobile robot navigation. In *Proceedings of the National Conference on Artificial Intelligence*, volume 2, pages 944–950. AAAI, 1996.

[246] S. Thrun, D. Fox, and W. Burgard. A probabilistic approach to concurrent mapping and localisation for mobile robots. *Machine Learning*, 31(5):29–55, 1998. Joint issue with Autonomous Robots.

[247] A. P. Tirumalai, B. G. Schunck, and R. C. Jain. Evidential reasoning for building environment maps. *IEEE Transactions on Systems, Man and Cybernetics*, 25(1), 1995.

[248] A. P. Tirumalai, B. G. Shunck, and R. C. Jain. Dynamic stereo with self-calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(12), 1992.

[249] J. K. Tsotsos. Behaviorist intelligence and the scaling problem. *Artificial Intelligence*, 72(2):135–160, January 1995.

[250] L. H. Tsoukalas, E. N. Houstis, and G. V. Jones. Neurofuzzy motion planners for intelligent robots. *Journal of Intelligent and Robotic Systems*, 19:399–356, 1997.

[251] I. Ulrich and J. Borenstein. Vhf+: Reliable obstacle avoidance for fast mobile robots. In *Proceeding of the 1998 IEEE International Conference on Robotics and Automation*, volume 2, pages 1572–1577, 1998.

[252] J. S. Wang, J. H. Zhang, K. Chen, Z. J. Fong, H. J. Zheng, and B. P. Zhang. The development of a multi-function walking machine with five legs. In *Proceedings of the 26th International Symposium on Industrial Robots*, pages 423–428, 1995.

[253] W. Wells III. Visual estimation of 3-d line segments from motion – a mobile robot vision system. *IEEE Transactions on Robotics and Automation*, 5(6):820–825, 1989.

[254] J. Weng, P. Cohen, and N. Rebibo. Motion and structure estimation from stereo image sequences. *IEEE Transactions on Robotics and Automation*, 8(3):362–382, 1992.

[255] G. Winkler. *Image analysis, random fields, and dynamic Monte Carlo methods : a mathematical introduction.* Springer-Verlag, 1995.

[256] G. Wyeth and I. Barron. An autonomous blimp. In *Proceedings of the International Conference on Field and Service Robotics*, pages 485–492, 1997.

[257] Y. Yaga, S. Fujimura, and M. Yachida. Route representation for mobile robot navigation by omnidirectional route panorama fourier transformation. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, volume 2, pages 1250–1255, 1998.

[258] B. Yamauchi and P. Langley. Palce recognition in dynamic environments. *Journal of Robotic Systems*, 14:107–120, 1997.

[259] B. Yamauchi, A. Shultz, and W. Adams. Mobile robot exploration and map-building with continuous localization. In *Proceedings of the 1998 IEEE/RSJ International Conference on Robotics and Automation*, volume 4, pages 3175–3720, 1998.

[260] B. Yamauchi, A. Shultz, and W. Adams. Integrating exploration and localization for mobile robots. *Adaptive Behaviour*, 1999. To appear.

[261] T. Yata, L. Kleeman, and S. Yuta. Fast-bearing measurement with a single ultrasonic transducer. *International Journal of Robotics Research*, 17(11):1202–1213, 1998.

[262] E. Yeh and D. J. Kriegman. Toward selecting and recognizing natural landmarks. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 47–53, 1995.

[263] J. Yen and N. Pfluger. A fuzzy logic based extension to payton and rosenblatt's command fusion method for mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(6):971–978, 1995.

[264] J. S. Zelek. Dynamic path planning. In *Proceedings of the 1995 IEEE International Conference on Systems, Man and Cybernetics*, volume 2, pages 1285–1290, 1995.

[265] A. Zelinksy. A mobile robot exploration algorithm. *IEEE Transactions on Robotics and Automation*, 8(2):707–717, 1992.

[266] Z. Zhang and O. Faugeras. A 3d world model builder with a mobile robot. *International Journal of Robotics Research*, 11(4):269–285, 1992.

[267] Z. Zhang, R. Weiss, and A. R. Hanson. Obstacle detection based on qualitative and quantitative 3d reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(1):15–25, 1997.

[268] Y. Zhao and S. L. BeMent. Fast boundary extraction for mobile robot sensor-detected images. In *Proceedings of the 5th IEEE International Symposium on Intelligent Control*, volume 2, pages 1050–1055, 1990.

[269] J. Y. Zheng, M. Barth, and S. Tsuji. Autonomous landmark selection for route recognition by a mobile robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 2004–2009, 1991.