Localisation and Navigation: Applying Biological Principles in Mobile Robotics

by

Robert Ollington, BSc. Hons.

Submitted in fulfilment of the requirements for the Degree of

Doctor of Philosophy

University of Tasmania (January, 2007)

Declaration

This thesis contains no material which has been accepted for a degree or diploma by the University or any other institution, except by way of background information duly acknowledged in the thesis.

To the best of my knowledge and belief, no material previously published or written by another person is included, except where due acknowledgment is made in the text of the thesis.

Robert Ollington

11 January, 2007

Abstract

Recently, there has been a significant effort to apply behavioural and anatomical studies of hippocampal place learning in rodents and other animals to the problem of robot localisation and mapping. The stated purpose of these recent experiments is twofold. Firstly, it is hoped that a study of this material will lead to improved algorithms for mobile robotics. Secondly, the behaviour of these new algorithms may be studied to evaluate psychological theories, and aid in the development of new theories. This thesis builds on these experiments by developing a complete localisation and navigational system for a simulated mobile robot. In order to provide a complete and efficient system, several new algorithms were developed.

Firstly, a method for preprocessing input was required, thus the adaptive response function neuron (ARFN) was developed. This neuronal model is able to identify similar input patterns, while discriminating between conceptually different sensory experiences. ARFNs learn a locally tuned response to input patterns, and are able to adapt the centre, width and shape of each input's response function on-line. These cells demonstrate one simple way that neurons in the cerebral cortex may learn a locally tuned response to input.

Secondly, a place cell system was developed for localisation. The new system provides a simple technique for establishing place cell firing based on odometric information and the current view (as captured by ARFNs). This system enables the robot's position to be accurately estimated, even in the presence of random and systematic odometric errors. The main advantage of the new system is that it allows certain topological assumptions to be made a priori, thus accelerating the training of downstream navigational systems. This prior knowledge may help explain the dead reckoning abilities of some animals and provides new insights into the place cell system in general.

Finally, a novel reinforcement learning algorithm was developed for goal independent navigation in complex environments. The new algorithm, called Concurrent Q-Learning (CQL), learns a value function for all goals simultaneously, and updates this value function more efficiently than similar algorithms. This is particularly true in dynamic environments, where CQL is shown to outperform other reinforcement learning algorithms. Unlike CQL, alternative methods for achieving goal-independent navigation, such as coordinate learning, cannot easily be applied to complex environments. Furthermore, the performance of CQL shows that

coordinate learning is not necessary to solve behavioural tasks previously thought to require an abstract vector representation.

While the focus of this research has been on spatial cognition, the hippocampus is also thought to be fundamental to other basic thought processes. Therefore, it is hoped that this research may stimulate further study not only into animal and robotic navigation, but also into biological and artificial intelligence in general.

Acknowledgments

I would like to thank Dr Peter Vamplew for his support and advice throughout the project. He has been a great friend and supervisor. Thanks also to my associate supervisor, Dr Ray Williams, for filling in for Pete, providing an alternate viewpoint, and for extensive assistance with other duties.

For their support and encouragement over many years, I would like to thank my parents; I would never have got this far without their help. Thanks also to the rest of my family, especially my wife Nadia for her patience, encouragement and valuable assistance throughout my studies, and to Isabel and Nicholas for putting up with my grumpy moods.

I would also like to thank: Richard Dazeley, for valuable discussions throughout the project; Sungsik Park, for his support and comments; and the G&AI research group, especially Adam Berry and David Benda, for help and distractions in equal measure. In addition, the entire academic staff of the School of Computing have been very supportive throughout my degree. In particular, I would like to thank Dr Julian Dermoudy and Nicole Clarke for feigning interest in my research, and for assistance with teaching and other duties.

For technical support, I also wish to acknowledge David Herbert, Terry Bigwood, Luke Fletcher, and the rest of the team. Also thanks to the administrative staff for their help.

Contents

Chapter 1. Introduction

Mobile robotics is an exciting field of study with applications in defence, exploration, accessibility, transportation and recreation. Mobile robots allow operations in areas that are unsafe, uninteresting or otherwise impractical for a direct human presence. Most of these applications require some level of autonomy. For example, a robot exploring the surface of Mars, cannot receive human guidance in real time and must be able to complete some tasks independently for efficient operation. Similarly, if a robot is required to perform a task that is considered uninteresting for a human operator, the robot must be able to act autonomously.

One key attribute required by mobile robots is the ability to localise and navigate within a potentially unfamiliar environment. While researchers have made dramatic improvements in this area in recent years, it is clearly evident that the navigational abilities of mobile robots are still easily outmatched by those of animals. It would seem apparent that a lot can be learned from studies of animal navigation. However, the study of such fundamental behaviour is not always easy.

Chomsky (1968, p24) stated that "one difficulty in the psychological sciences lies in the familiarity of the phenomena with which they deal". This statement is equally true for the field of artificial intelligence, a field closely associated with the psychological sciences. For abilities involving spatial cognition and navigation, this is especially true. Even simple questions such as "how do I know where I am?" can be very difficult to answer either informally or formally. Despite these difficulties, we are able to apply these cognitive abilities with ease to solve complex spatial problems.

While familiarity remains a problem, it is somewhat easier to analyse the behaviour of other more primitive animals that also display remarkable navigational abilities. Tolman (1948) reviewed many experiments dealing with the navigational abilities of rats. It was shown that rats were able to learn the layout of a complex maze-like environment. Despite many similar views in this environment, the rats were able to find their way to a food source, taking fewer wrong turns with each successive trial. In addition, the rats were able to learn about the environment even in the absence of any reward. When a reward was later added to the environment, the rats were able to use this latent learning to return directly to the location of the reward. The ability of rats to reason a shorter path to the goal was also demonstrated. Rats were trained to follow a restricted path to the goal. When that path was later blocked and several

new paths opened, the rats were able to choose the path that lead most directly to the goal location. Similar abilities have been documented for many other animal species, ranging from ants (Wehner & Räber, 1979) and bees (Dyer, 1996), to birds (Wiltschko, 1997) and other rodents (Alyan & Jander, 1994; Etienne, 1987; Mittelstaedt & Mittelstaedt, 1980).

In contrast to the ease with which animals are able to solve complex navigational tasks, traditional techniques in artificial intelligence have difficulty solving some problems that appear relatively simple. While this is most apparent for simple sensory processing, it is also true of higher cognitive processes such as spatial cognition. It is therefore important to gain a greater understanding of the biological mechanisms in order to develop improved artificial navigation algorithms. Conversely, Hirtle and Heidorn (1993) have stated that the development of a computational model may aid in the development of biological theories by focusing on the processes and representations involved.

The aim of this thesis is to develop an artificial navigation system based on studies of animal navigation and biology, with the goals of extending the range of tools available for use in mobile robotics, and to gain a better understanding of the biological systems. While this is not the first experimental work in this area, previous studies have focused mainly on localisation and mapping and have not explored the relationships between these systems and navigation. Here a holistic approach is taken covering localisation and both low level and high level navigational cognition.

1.1. Hypotheses

Given that animals display navigational abilities that clearly outmatch those of mobile robots, it was hypothesised that:

> *A study of past and recent psychological and anatomical studies may lead to new navigational solutions that may be applied in the field of mobile robotics. In particular, a navigational system developed in this way should be able to deal gracefully with dynamic goals and environments, and produce apparently natural behaviour in the face of uncertain or incomplete information.*

Furthermore, it was hypothesised that:

> *The implementation of a biologically inspired solution for localisation and navigation may provide valuable new insights in the field of spatial cognition. This should be particularly true for interactions between the localisation and navigational system, as this is an area that has not been extensively studied.*

Finally, this research may have relevance to fields other than spatial cognition. The brain areas associated with localisation and navigation also play major, and presumably similar, roles in other cognitive tasks. Therefore, it should be possible to adapt algorithms based on these biological systems to more general problems in the field of artificial intelligence.

1.2. Methodology

To assess the validity of these hypotheses it was proposed that a complete navigational system, based on developments in the field of spatial cognition, be developed for a simulated mobile robot. While experiments conducted in simulation only will not provide a definitive verification of the proposed methods, there are many advantages of such an approach. Aside from time and cost, simulated experiments allow a range of environments and robot configurations to be tested quickly, and allow the researcher to concentrate on the algorithms rather than the hardware.

Given that in nature successful navigation is not reliant upon a well-developed visual system, it was decided to implement the system for a simulated robot with range, tactile, and odometric sensors only. A full description of the simulated robot and environment can be found in Appendix A.

In addition to this simulation, more simplistic problems were also considered in the development of some algorithms. For example, classification problems were used in Chapter 5, and grid-world problems were used extensively in Chapter 8.

1.3. Structure of the Thesis

Chapter 2 reviews the literature regarding biological mechanisms for localisation and navigation, while Chapter 3 examines some previous computational models that implement these theories in simulated and real mobile robots. Chapter 4 presents the design of the proposed model and relates this to previous models.

Chapter 5 and Chapter 6 discuss the implementation of systems used for localisation. Chapter 7 discusses a system for low-level navigation, and in Chapter 8 a novel reinforcement learning algorithm for path planning is developed. Chapter 9 details the integration of these sub-systems and presents the results of testing in various configurations and environments.

Finally, Chapter 10 concludes by relating the findings to previous and future work. Appendices are also included to provide detail of the simulation environment, a summary of symbols and notation, and a list of publications arising from the thesis.

Chapter 2. Localisation and Navigation in Nature

Animals and humans show a remarkable ability to navigate in complex environments with apparent ease. This ability can be broken down into a number of non-trivial sub-tasks. These tasks include:

Localisation. The ability to know one's current location and orientation with respect to the environment. While this ability may seem trivial, it is in fact a complex task requiring the interaction of many sensory systems.

Path Integration. Also known as dead reckoning, this ability allows an animal to track its progress as it moves around an environment. If an animal wishes to return to a previous location, path integration allows a direct route to be calculated. Path integration is also a critical component of localisation.

Mapping. Many navigational tasks require some form of spatial map to be learned and committed to memory. While some navigational tasks may be performed through a simple sensor/action association (*taxon*[1] navigation), many tasks require a more abstract representation of ones environment.

Path Planning. Even with a map of the environment, path planning can be a difficult task in many environments. Furthermore, a robust path planning system should include the ability to find detours around novel obstructions, and to find shortcuts as these become available.

Goal Identification. While the identification of some goals may be quite straightforward, others can be more complex. Goal identification not only needs to identify important locations related to such primary needs as food and shelter, but also needs to address the issues of exploration and threat avoidance.

This chapter investigates some of the biological mechanisms that underlie these abilities. Section 2.1 introduces the concept of cognitive maps, and reviews evidence that the hippocampus may be the locus of this mapping ability and other aspects of spatial cognition. Section 2.2 examines information input to the

---

[1] Taxon navigation is the term used to describe the group of navigational strategies based on simple stimulus-response-stimulus or route-like algorithms (O'Keefe & Nadel, 1978).

hippocampus and section 2.3 discusses some theories of how this input, along with a system for path integration, may provide a means for localisation. Section 2.4 presents new evidence suggesting that navigation and path planning may be achieved through reinforcement learning in the basal ganglia. Finally, the main points are summarised in section 2.5.

2.1. Cognitive Maps and the Hippocampus

Tolman (1948) proposed that the brain might hold a topological map of its environment, and that this map could be used for various navigational tasks. This cognitive map theory has also been strengthened by later experiments, such as those involving the Morris Watermaze (Morris, 1981; Steele & Morris, 1999).

The Morris Watermaze (Morris, 1981) is an example of a problem that cannot be solved without an abstract representation of space (Muller, Kubie, Bostock, Taube, & Quirk, 1991). The Watermaze consists of a cylindrical environment filled with an opaque liquid. A platform is placed just below the level of the liquid so that it cannot be seen by a swimming rat.

In the reference memory in the watermaze (RMW) task (Foster, Morris, & Dayan, 2000; Morris, 1981; Steele & Morris, 1999) rats are trained to find the location of the hidden platform over a period of several days, undergoing four trials per day. After this initial training period, the platform is moved to a new location. Once the new platform location is discovered, the rats are able to navigate directly to the new location on subsequent trials.

In the delayed matching-to-place (DMP) task (Foster et al., 2000; Steele & Morris, 1999) the platform is moved at the end of every day. Even in this more complex task, the rats are able to achieve "one-trial learning"[2] after very few days. Typical results for the RMW and DMP tasks are shown in Figure 2.1.

[2] One-trial learning is the ability to immediately repeat a task after one successful trial.

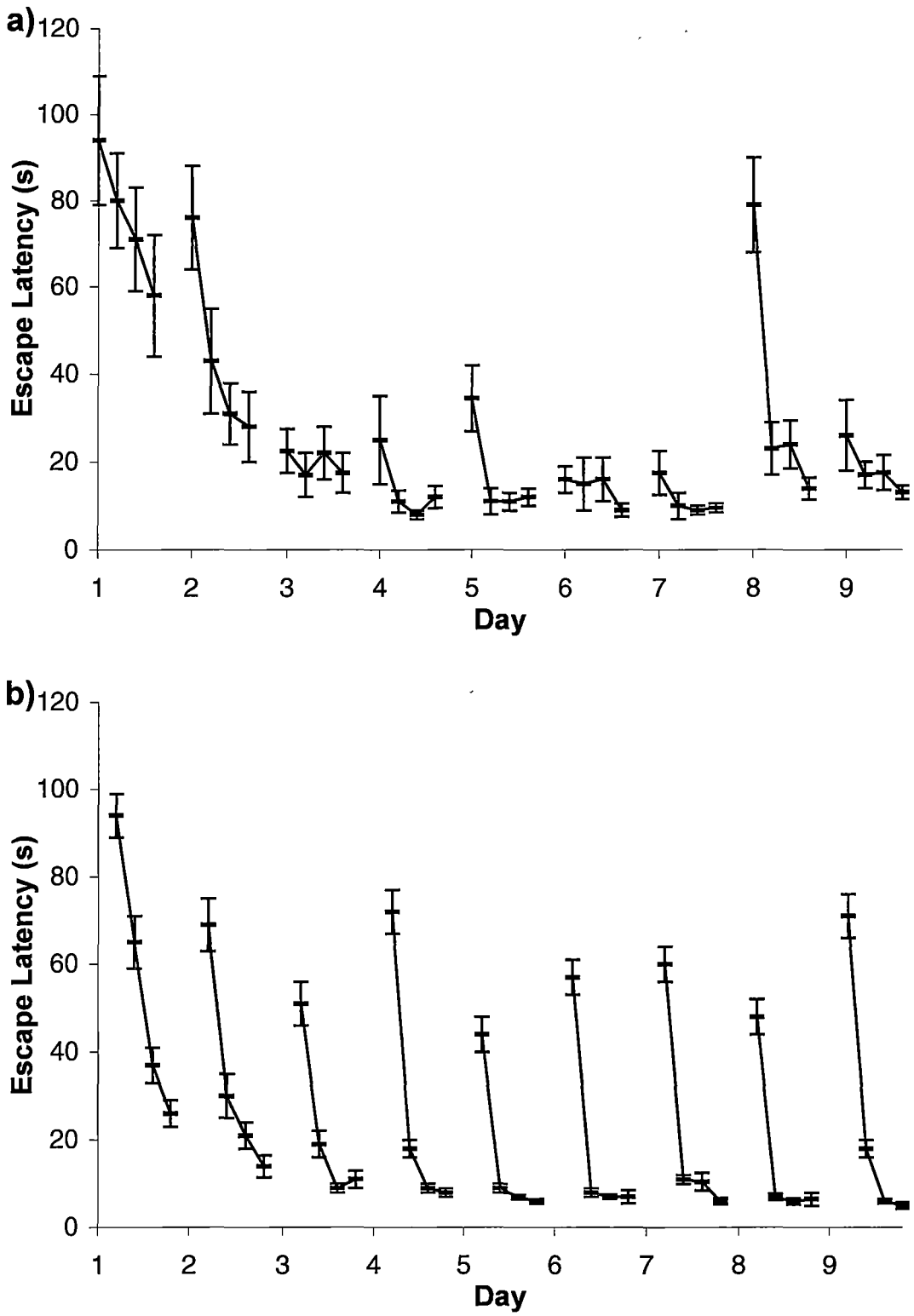

Figure 2.1 Performance of rats in the Morris Watermaze in a) the Reference Memory in the Watermaze, with the platform moved on day 8, and b) the Delayed Matching-to-Place tasks, with the platform moved at the beginning of each day. (reproduced from Morris, 1981; Steele & Morris, 1999)

Studies of brain lesions in animals (see Barnes, 1988 for a review) and humans (Habib & Sirigu, 1987) have identified the hippocampus as a possible location for the cognitive map proposed by Tolman. Figure 2.2 shows the hippocampus including some of the key neural connections. The hippocampus consists of two thin layers of neurons, called the dentate gyrus and Ammon's horn (*cornu Ammonis*, abbrev. CA), that are folded over each other. Ammon's horn is divided into several groups of neurons of which only CA1 and CA3 are relevant to this discussion. The hippocampus receives most input from the entorhinal cortex via a group of axons called the perforant path. Perforant path axons synapse on granule cells in the dentate gyrus, which in turn form connections with pyramid cells in CA3. CA3 neurons send output from the hippocampus via the fornix, to neurons in CA1 via the Schaffer collateral, and also to a very large number of other CA3 neurons. CA1 output also departs the hippocampus via the fornix, and the subiculum, which sends output back to the entorhinal cortex, thus completing a circuit.

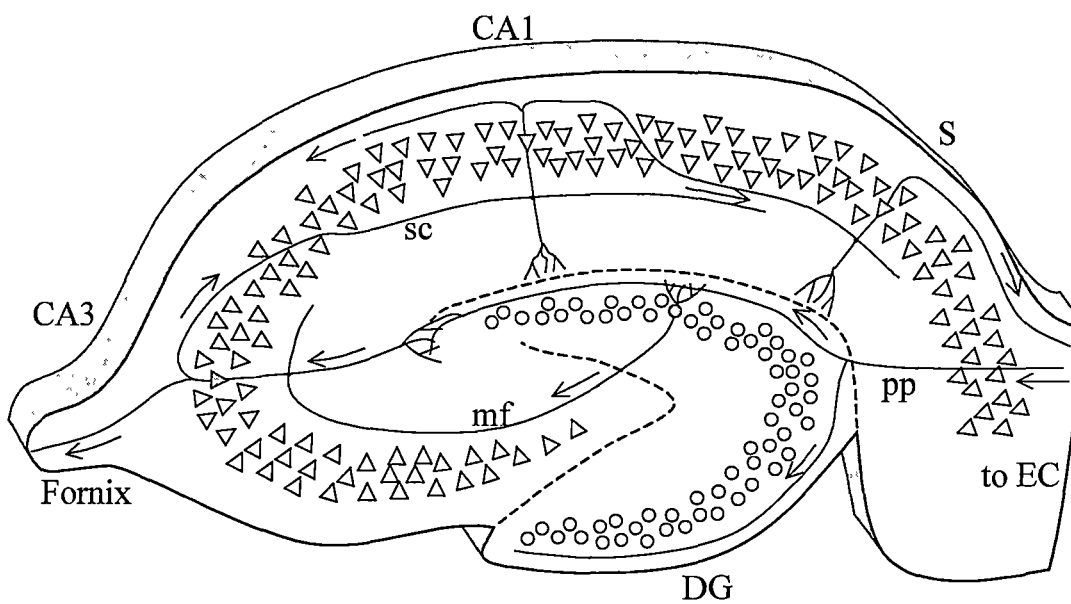

Figure 2.2 The hippocampus, including some neural connections. Axons comprising the perforant path (pp), arising in the entorhinal cortex (EC), make connections with granule cells (small circles) in the dentate gyrus (DG). Mossy fibre (mf) projections from DG make strong connections with pyramid neurons (small triangles) in CA3. These neurons send output via the Schaffer collateral (sc) to CA1, also exiting the hippocampus via the fornix. Neurons in CA1 send the majority of their output to the subiculum (S) that in turn send output back to the entorhinal cortex. (adapted from Amaral & Witter, 1989)

O'Keefe and Dostrovsky (1971) observed that pyramid cells in the hippocampus of rats responded maximally when the rat was in a certain location. The region in the environment where a place cell, as these neurons are now known, fires most strongly is known as the cell's place field. The properties of place cells and place fields include:

- Place fields are established within about ten minutes of entering a new environment (Wilson & McNaughton, 1993).

- Place fields tend to follow local barriers within the environment. For example, Muller and Kubie (1987) found place fields in a cylindrical environment that extended along the wall of the cylinder, with the interior edges of these fields being concave.

- The combined output from a relatively small group of place cells is sufficient to accurately predict the rat's position to within a few centimetres (Wilson & McNaughton, 1993). The combined output of all place cells is often referred to as the place code.

- Place fields are influenced by visual stimulus. If visible landmarks within an environment are rotated, place fields rotate with respect to each other by the same amount (Muller & Kubie, 1987; O'Keefe & Speakman, 1987).

- In the absence of visual stimulus, place cells persist (Muller & Kubie, 1987; O'Keefe, 1976; O'Keefe & Speakman, 1987). Hence idiothetic information, such as vestibular, visual motion and motor efferent inputs, must also be able to influence place cell firing (Bures et al., 1999). Other experiments also confirm that path-integration or dead-reckoning is a crucial component of navigation in many animals (Alyan & Jander, 1994; Etienne, 1987; Mittelstaedt & Mittelstaedt, 1980).

- Some place cells show correlations to non-spatial aspects of the environment, and it has been suggested that these cells may code for context with space being just one of the relevant parameters (Eichenbaum, 1996; Eichenbaum & Cohen, 1988; Eichenbaum, Otto, & Cohen, 1992; Markus et al., 1995; Muller & Kubie, 1987). For example, some cells show a correlation with the current behaviour of the rat.

- Place cell activity is independent of goal location (Speakman & O'Keefe, 1990).

- The proximity of place cells in the hippocampus bears no correspondence with the proximity of their place fields within the environment (Muller & Kubie, 1987; O'Keefe, 1976).

- Place fields in different environments are not correlated, and a cell exhibiting a place field in one environment may have no place field in another (Muller & Kubie, 1987).

- Place cell firing actually predicts the future position of the rat on a short time-scale (~100ms) (Muller & Kubie, 1989).

- Place cells have been found in other brain areas, but those in areas CA1 and especially CA3, are most correlated with the rat's location (Amaral & Witter, 1989).

O'Keefe and Nadel (1978) suggested that these place cells might form the basis of a system for localisation and navigation. They proposed two different mechanisms; a "taxon" system, and a "locale" system. The taxon system was used for route learning. For a given route, each place cell would be associated with an appropriate response leading to the next location on the route. The locale system could be used for map-like navigation. The map was proposed to be an absolute Euclidean representation of the environment (O'Keefe, 1989, 1990, 1991). Such a representation would allow distances and directions to be calculated between the field centres of place cells.

Having established the existence of place cells, and having proposed that these cells form part of a cognitive map of the environment, it is natural to ask how place cell firing arises, and hence how an animal may localise within its environment. For this it is necessary to understand how the hippocampus interacts with other brain areas, and in particular what inputs the hippocampus receives. Figure 2.3 shows some of these interactions. The following sections discuss the implications of this network of connections, and in particular considers hippocampal input, the formation of the place code, and hippocampal output and its possible influences on navigation.

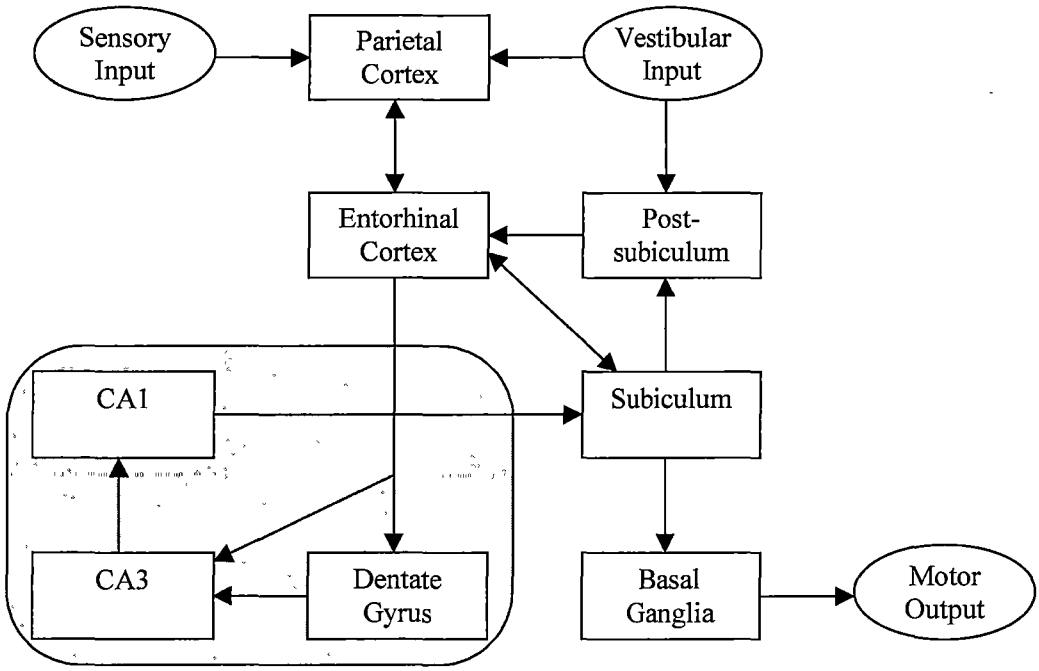

Figure 2.3 Some of the functional connections of the hippocampus and the place cell system.

2.2. Hippocampal Input: Head Direction and Local View

The major source of input to the hippocampus is the entorhinal cortex, and while some connections are made with areas CA1 and CA3, the majority of this input is to the dentate gyrus. The entorhinal cortex receives highly processed sensory information originating in the parietal cortex (Deacon, Eichenbaum, Rosenberg, & Eckmann, 1983), which receives sensory input including visual and vestibular input. The parietal cortex also receives feedback from the entorhinal cortex. The postsubiculum receives vestibular input and is a source of input to the entorhinal cortex. The subiculum is also a major source of input to the entorhinal cortex.

Cells with some correlation to place have been found in the parietal cortex, entorhinal cortex and subiculum. These cells are generally not referred to as place cells since the correlation is coarse and noisy. Cells that are highly correlated with head direction, irrespective of location, have been found in the postsubiculum. These are referred to as head direction cells (Ranck, 1984; Taube, Muller, & Ranck, 1990). Some head direction cells have also been found in the subiculum. Cells whose firing is correlated with both place and direction have also been found in the subiculum (Sharp & Green, 1994).

2.2.1. Head direction

Orientation and location are two interacting concepts necessary for absolute localisation[3], with orientation being perhaps the simpler concept (Muller et al., 1991). It seems sensible then, to examine the head direction system before attempting a detailed analysis of the place cell system.

Cells have been found in the postsubiculum that fire only when the rat's head is oriented in a particular direction. These head direction cells have many properties in common with place cells:

- The firing of head direction cells is independent of behaviour.
- The population of head direction cells provides an accurate, distributed representation for any head direction (Blair, Lipscomb, & Sharp, 1997).
- The firing of head direction cells is maintained even in total darkness (McNaughton, Chen, & Markus, 1991)
- Local landmarks influence the firing of head direction cells and may be used to correct errors in the head direction signal (McNaughton, Markus, Wilson, & Knierim, 1993; Taube & Burton, 1995; Taube et al., 1990).

Other closely related brain areas also contain cells that clearly play an important role in maintaining the head direction signal. Neurons correlated with angular head velocity have been found in the dorsal tegmental nucleus (Basset & Taube, 2001), and in the anterior thalamus, head direction cells have been found that predict the rat's future head direction (Blair & Sharp, 1995). Head direction cells that fire more strongly when the rat is turning have been found in the lateral mamillary nucleus (Leonhard, Stackman, & Taube, 1996). Also of interest is the fact that the tuning curves (a plot of firing rate versus direction) of head direction cells are often distorted when the animal rotates as shown in Figure 2.4.

---

[3] Absolute localisation is the ability to localise immediately upon entering an environment. Incremental localisation is the ability to maintain a position estimate during navigation.

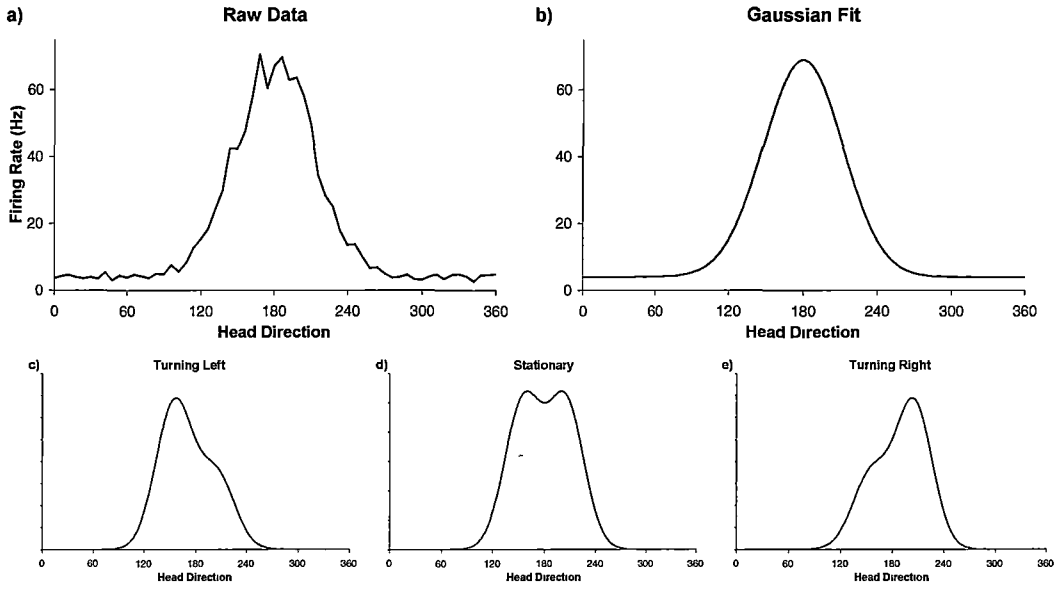

Figure 2.4: Head direction cell tuning curves. a) Typical raw data for the tuning curves of a uni-modal cell in the anterior thalamus. b) Tuning curves are normally approximated to a Gaussian fit. c), d) and e) show idealised tuning curve distortion for an animal turning to the left and right for a cell with two tuning curve peaks. (adapted from Blair et al., 1997; Goodridge & Touretzky, 2000)

A number of models have been proposed to explain the firing of head direction cells, with the majority of researchers agreeing with the basic principles. The proposed cell populations and representative connections are shown in Figure 2.5. Not shown are connections with cells encoding visual input; these would presumably make connections with head direction or turn-modulated head direction cells. It is expected that visual information, such as the relative direction to distal landmarks, would become correlated with head direction over time to facilitate the correction of integration errors. With the addition of a place signal, local landmarks could also be used to correct for errors in head direction integration.

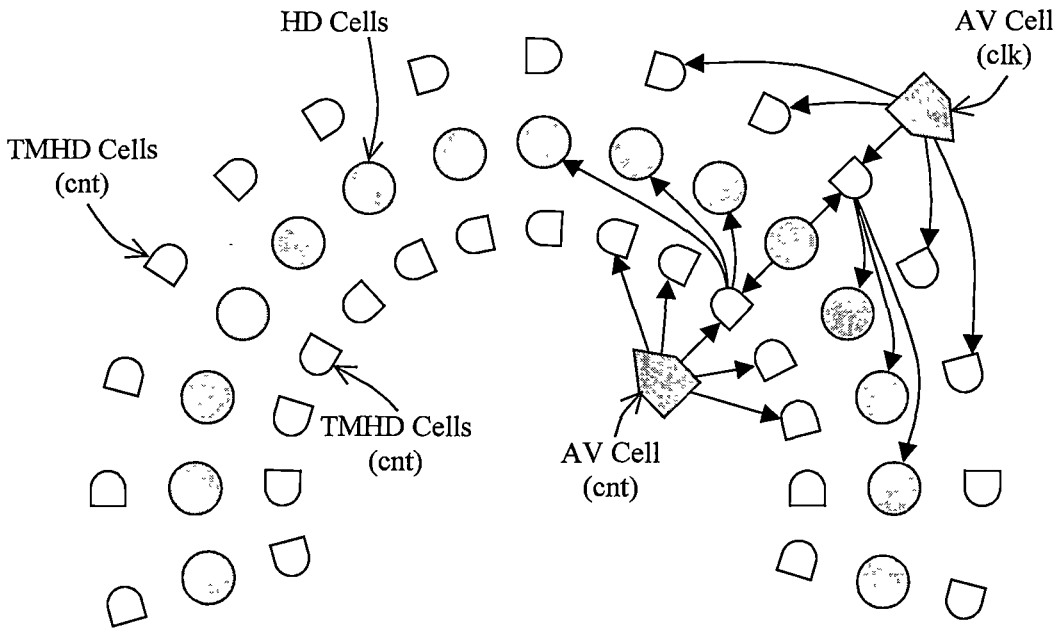

Figure 2.5: Model of the head direction circuit: showing head direction (HD), angular velocity (AV), turn-modulated head direction (TMHD) cells, and representative connections. Separate populations of AV and THMD cells are proposed for clockwise (clk) and counter-clockwise (cnt) rotations. Each HD cell excites neighbouring TMHD cells, which in turn excite neighbouring HD cells in each direction. When the animal is not turning THMD input to HD cells is uniform in each direction, but when the animal turns AV cells increase the firing of corresponding THMD cell populations. This asymmetric input causes the activity of HD cells to shift in the appropriate direction. (adapted from Blair et al., 1997)

McNaughton and colleagues (1991) suggest that the integration of angular head velocity information is accomplished using a simple look-up table approach. Given the current head direction and the current angular velocity the conceptual table would store the unique head direction that would result after a certain time delta. The table would presumably be implemented via TMHD→HD cell connections.

The attractor model of the head direction system was later developed to further explain the dynamics of the system (Skaggs, Knierim, Kudrimoti, & McNaughton, 1995). An attractor is a neural network with a pattern of connection weights such that only a small number of possible network states are stable. Any non-stable state will quickly be transformed into a stable state through the intrinsic dynamics of the system. A one-dimensional ring attractor can be constructed such that the only stable states are those with a 'hill' of activity at a certain position on the ring (Zhang, 1996). By providing external input to the left or right of the active neurons, the 'hill'

can be made to move in that direction. Through the choice of appropriate parameters, a network can be constructed that integrates angular velocity quite accurately.

A further refinement of the attractor hypothesis was developed that involved the coupling of two attractor networks representing cell populations in the anterior thalamus and postsubiculum respectively (Redish, Elga, & Touretzky, 1996). A similar model was later developed by Goodridge and Touretzky (2000) that also accounted for deformation of head direction tuning curves in the anterior thalamus (see Figure 2.4).

2.2.2. Local View

Place cells are strongly influenced by the local view, and it has been suggested that the source of this local view information is the entorhinal cortex (Redish & Touretzky, 1997). The assertion that entorhinal cortex cells are directly associated with hippocampal place cells is strongly supported by the fact that the effects of cue rotation and removal on place cell firing is mirrored in the firing of entorhinal cortex cells.

The entorhinal cortex receives highly processed sensory information from neocortical areas, and entorhinal cortex 'place' cells are more influenced by sensory information than true place cells (Muller et al., 1991). Unlike place cells in CA3 and CA1, cells in the entorhinal cortex generally have 'place' fields in all environments (Muller et al., 1991), further supporting the notion that these cells may essentially form a coding for local view.

If entorhinal cortex cells do code for local view, then that view must be in allocentric[4] coordinates, since the firing of these cells is independent of the current head direction. In order to convert egocentric sensory information into an allocentric view, the entorhinal cortex must receive information about the current head direction. The entorhinal cortex does receive input from the postsubiculum and this input is likely to include head direction information, further supporting the local view hypothesis.

---

[4] Strictly speaking, allocentric refers to an environment based reference frame, or world-centred coordinates. However as in this case, it is often used to describe coordinates centred on the current animal location but with orientations relative to the environment. Egocentric refers to an animal centred reference frame.

Since place cells are more sensitive to changes in the local environment than to changes in distal landmarks (Muller & Kubie, 1987), it seems likely that any view cells influencing the firing of place cells will also be more sensitive to local cues. In particular, the distance to and orientation of nearby walls seems to have a particularly strong effect on place fields, and hence should be a major factor in view cell firing.

2.3. Place Cell Learning: Path Integration and Localisation

The main input to the hippocampus comes from the entorhinal cortex, and it has been proposed that the function of some entorhinal cortex cells is to identify local views. While hippocampal place cells are influenced by visual sensory cues, they also continue to fire in complete darkness, suggesting that local view cells are not the only influence on the firing of hippocampal place cells. In the absence of sensory cues, the only explanation is that the animal localises through some form of path integration or dead reckoning (McNaughton et al., 1991; Muller et al., 1991; O'Keefe, 1976). Evidence for path integration can be seen in the ability of a wide range of animals to return to a starting location after taking a circuitous route, even in total darkness (Alyan & Jander, 1994; Etienne, 1987; Mittelstaedt & Mittelstaedt, 1980). Furthermore, Sharp and colleagues (1995) report that hippocampal place cells are influenced by vestibular and visual motion inputs.

The functioning of the path integrator would be analogous to the head direction system described earlier. Input representing the perceived self-motion of the animal would move the centre of activity of the integrator cells. In this case, these would conceptually (but not necessarily physically) be arranged in a two-dimensional array. Input from local view cells would then allow corrections to be made to adjust for errors in the self-motion input.

It has been suggested that hippocampal place cells themselves form the basis of a path-integration system rather than a topological map (McNaughton et al., 1996). It is suggested that the ten minutes required for stable place fields to develop (Wilson & McNaughton, 1993) would not be enough time for the formation of a consistent topological map. However, since path-integrator connections could be largely pre-configured, ten minutes should be ample time to establish place fields of a path integrator. The only learning required would be the binding of local view and place cells to correct for integration error. It has been shown that the correlation of place fields in a novel environment can be predicted by previous activity correlations of

place cells during sleep (Kudrimoti, McNaughton, Barnes, & Skaggs, 1995). The ability to make such a prediction suggests some degree of pre-configuration.

McNaughton and colleagues suggest this path integration system would operate in a similar way to their model for head direction integration (McNaughton et al., 1991). As with the head direction model, cells that are correlated with position and the direction of movement should have a direct influence on place cells. Cells in the subiculum satisfy this requirement but have only an indirect influence, via the entorhinal cortex, on place cells in the hippocampus. In support of this Redish and Touretzky (1997) propose that path integration is performed by a loop consisting of the hippocampus, subiculum and entorhinal cortex. In their model, local view and path integrator input is combined in the dentate gyrus, and these cells drive the place cells of CA3 and CA1. If either the path integrator or local view input changes a different hippocampal place cell will be activated (Redish & Touretzky, 1999). Output from these place cells then feeds back to the path integration circuit via the subiculum.

In a similar way to the head direction system, path integration may be accomplished in part by an attractor network (Kali & Dayan, 2000). Recurrent connections between cells in CA3 could form the basis of a two-dimensional attractor network with a hill of activation representing the location of the animal in the environment. Applying appropriate self-motion related input could shift the hill of activation to facilitate path integration.

2.4. Hippocampal Output: Path Planning and Goals

The major output from the hippocampus is from CA1 to the subiculum. The subiculum sends output to the basal ganglia, which in turn sends output to areas associated with motor control. Reinforcement learning is a major function of the basal ganglia. The firing of dopaminergic neurons in the basal ganglia is highly correlated with the error in reward prediction (Schultz & Dickinson, 2000; Schultz, Tremblay, & Hollerman, 2000). Neurons in the striatum show correlations to expected and experienced rewards and also to the initiation and execution of actions related to those rewards (Schultz & Dickinson, 2000; Schultz et al., 2000). Suri (2002) proposes that the actor-critic temporal difference learning architecture (Barto, Sutton, & Anderson, 1983) is a suitable model for some functions of the basal ganglia. An overview of the architecture is shown in Figure 2.6.

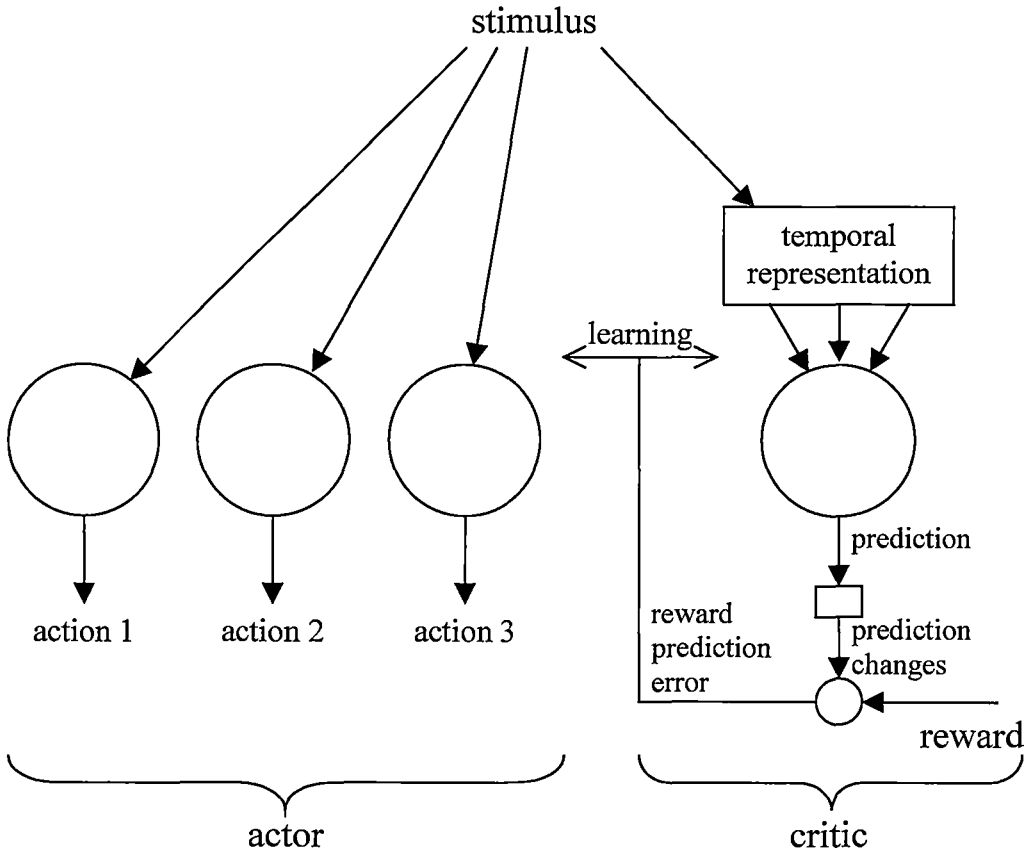

Figure 2.6: Neural implementation of actor-critic temporal difference learning. (adapted from Suri, 2002)

In Figure 2.6, the critic learns to predict the value of the current state, as represented by the input stimulus. The small circle in the figure represents dopamine neurons, which calculate the error in the predicted value. This error is then used to train the critic, and also the actor. The actor consists of discrete units corresponding to each possible action. The output of each of these actor units is a measure of the suitability of performing that action given the current stimulus. See section 7.1 for a full description of temporal difference learning and the actor-critic architecture.

Little work has been conducted to fully examine the connection between navigation and the basal ganglia. However, initial investigations strongly suggest that reinforcement learning may play a significant role in navigation. For example, Mura and Feldon (2003) showed that lesions of the dopaminergic system in rats led to a profound deficit in the ability of the animal to find the platform in the Morris watermaze. Combined with the theory that the dopaminergic system is closely related to reinforcement learning, this suggests that reinforcement learning may play an important role in this learning task.

Another indicator that the basal ganglia are associated with reinforcement learning and navigation, arises from interactions with the hypothalamus. The hypothalamus is a centre for controlling motivational states (Swanson & Mogenson, 1981) and this motivational information is sent to, among other brain areas, the basal ganglia. Since satisfying many motivations will require moving to a particular location (e.g. moving to food), it seems likely that motivational signals would be sent to an area of the brain involved with navigation. This concept is explored further by Guazzelli, Arbib and colleagues (Arbib, 1999; Guazzelli, Corbacho, Bota, & Arbib, 1998) through their world graph theory. They propose a model for determining the rewards of an actor-critic learning algorithm by considering the current set of motivational drives. Neurons encoding the current motivational drives are assumed to reside in the hypothalamus, the output of these neurons then influences the firing of dopamine neurons in the basal ganglia.

Brown and Sharp (1995) developed what is essentially a reinforcement learning model of navigation by considering the interaction of place and head-direction cells, and motor neurons in the nucleus accumbens. In the model, the activity of place cells and head-direction cells result in the firing of cells in either of two groups of motor cells. One group corresponds to moving left and the other to moving right. A trace is kept of which group of cells fire for a given place and head-direction cell combination, and this trace decays over time. When the goal is encountered, synaptic connections between place and head-direction cells and motor neurons are strengthened according to the corresponding trace.

2.5. Summary

This chapter has reviewed the current understanding of spatial cognition with emphasis upon the concept of cognitive maps. Hippocampal place cells have been identified as an important component of the localisation and mapping system. Place cell firing is maintained through input from the visual processing system and head direction cells in the postsubiculum. A method for path integration has also been cited as a crucial component of the system and, while further work is needed, this appears to be accomplished via a loop consisting of the hippocampus, subiculum, postsubiculum, and entorhinal cortex. Finally, it has been suggested that the basal ganglia also play an important role in navigation and path planning and that a reinforcement learning approach may be the most appropriate method for modelling this type of learning.

Chapter 3. Localisation and Navigation: Computational Models

The field of mobile robotics is large and diverse. It would not be possible to review all of the research in the field relating to localisation, mapping and navigation, and furthermore much of this information would not be relevant to this thesis. The main aim of the thesis is to examine biological mechanisms that may be useful in the field of mobile robotics. Therefore, this chapter will review those computational models that demonstrate applicability to mobile robotics and that claim some degree of biological inspiration. In particular, those models inspired by the mammalian place cell system described in the previous chapter will be reviewed.

Section 3.1 will review models of localisation and mapping that attempt to simulate the place cell system itself. In section 3.2, navigational models utilising a place cell representation will be discussed. Section 3.4 will summarise this literature and discuss the strengths and weaknesses of experimental work to date.

3.1. Localisation and Mapping

3.1.1. Analysing the Local View: Extracting Landmarks

The first stage of localisation in biological systems is the activation of view-cells. Likewise, for all of the computational models reviewed, processing the current view formed an important first step in the localisation procedure.

Typically landmarks are first extracted from the local sensory view. The type, bearing, or range of each landmark (or some combination of these) is then either further processed or passed directly to the place cells. While this general principle is common for most of the systems reviewed, they differ in the type of sensory information provided, the landmark information that is used, and the degree of further processing of this information.

In the simplest case, Guazzelli, Bota and Arbib (2001) conducted experiments in simulation only with the bearing and distance of three distal landmarks given directly to the agent.

Burgess, Donnet and O'Keefe (1996; 1998) show that it is possible to extract landmarks from a real, though simple, environment with experiments conducted on a

Khepera[5] mobile robot. The robot sensors consisted of video and short-range (4cm) infra-red proximity sensors. The environment was a rectangular 'room' with white walls and a dark floor. One wall had an identifying dark strip. As in the work of Gauzzelli and colleagues (2001), the range and bearing of landmarks, which in this case were the four walls of the environment, were used. The landmarks were found by rotating the robot to face each wall and acquiring an image. The image was then analysed to find the centroid of each wall, from which range and bearing information could be calculated.

Gaussier and colleagues (Gaussier, Joulain, Banquet, Lepretre, & Revel, 2000; Gaussier, Revel, Banquet, & Babeau, 2002) also extracted landmarks from camera images, however their system demonstrates that it is possible to extract useful landmarks in a more natural environment. The system was implemented on a Koala[6] mobile robot equipped with a video camera capable of taking panoramic images over a 300 degree range. The robot operated in a 7.3m x 5.4m laboratory environment. Landmarks were extracted from the camera image using pattern recognition techniques. The image was first scanned for points of interest indicated by changes in horizontal image intensity. The area around these focal points was then compared to learned views. As with the two models discussed above, the bearing of each of these landmarks was extracted, but in contrast the type of landmark was used rather than the range.

Wan, Touretzky and Redish (Touretzky, Wan, & Redish, 1994; Wan, Touretzky, & Redish, 1994a, 1994b) show that landmarks can also be extracted in real environments from more rudimentary sensory input. Their model was implemented on a Xavier[7] mobile robot. This robot was equipped with a ring of 24 sonar sensors, an infrared laser rangefinder, and a colour camera. Information from the sonar sensors was stored in an occupancy grid[8] and standard edge detection algorithms were used to detect corners. The locations and types (concave or convex) of these

---

[5] Khepera is a small mobile robotics platform. See www.k-team.com for details.

[6] Koala is a mid-sized mobile robotics platform. See www.k-team.com for details.

[7] Xavier is another mid-sized robot. See www-2.cs.cmu.edu/~Xavier for details.

[8] An occupancy grid, also called a free space map, divides the space into discrete cells and labels each cell as either occupied or unoccupied. The occupancy may also be a fuzzy value.

corners become the landmarks of the system. Ranges and bearings to these landmarks were used along with the angle of incidence between the landmarks.

The amount of further processing conducted on the extracted landmark information varies between researchers. Gaussier and colleagues (2000; 2002); and Wan, Touretzky and Redish (Touretzky et al., 1994; Wan et al., 1994a, 1994b) perform no additional processing, the raw landmark information is used directly as input to the place cells. The remaining models discussed in this section use the landmark information as input to a view cell layer where the information is refined before being sent to the place cell layer.

The view cells in the model of Burgess, Donnet and O'Keefe fired maximally when a particular wall was at a set distance from the robot. The output of the sensory cells was calculated using a Gaussian function, with the width of the Gaussian modified by the preferred distance of the wall, increasing as the preferred distance increases. Equation 3.1 gives the activation function for the i^{th} sensory cell, where x is the distance from the wall, d_i is the cell's preferred direction, and A and σ are tuning parameters.

$$\frac{A}{\sqrt{2\pi\sigma^2 d_i}} \exp\left[\frac{-(x-d_i)^2}{2\sigma^2 d_i}\right] \qquad \textbf{3.1}$$

Guazzelli, Bota and Arbib (2001) first form view cells that respond to the bearing and range of one particular landmark. A further layer of cells then receives input from a selection of the primary view cells corresponding to different landmarks.

In contrast to all of the models discussed so far, Arleo and Gerstner (2000) did not explicitly extract landmarks from sensory information. Experiments were conducted using a Khepera mobile robot in a 60×60cm square environment surrounded by walls painted with vertical black and white stripes of various widths (barcode style). Features were first extracted from a video image of the walls in the current heading by using Walsh-like filters (Andrews, 1970). They defined five classes of filters each corresponding to a different one-dimensional horizontal pattern. From these classes they then defined ten filters corresponding to different frequencies, enabling a degree of range discrimination (the same pattern at a greater range will have a higher frequency). While this is not a landmark-based system, it could be argued that each view cell is responding to the presence of a particular landmark at a particular range. As with the model of Guazzelli and colleagues (2001), higher-level

view cells were then generated that depend on the activation of several simple view cells.

3.1.2. Generating Place Cells from the Local View

Burgess, Donnet and O'Keefe (1996; 1998) and Gaussier, Revel and Banquet (2000; 2002) each demonstrate that it is possible to generate simulated place cells that exhibit many of the properties of their biological equivalents from landmarks alone.

In the model of Burgess and colleagues (1996; 1998), the view cell output is sent to the next layer of cells, which model cells in the entorhinal cortex, via hard-wired connections. Each of these cells receives input from two view cells responding to two orthogonal walls. Output from this layer goes to the place cell layer; these on/off connections are trained using a form of competitive learning. Place cells then send output to goal cells, presumed to be in the subiculum. The structure of the model is shown in Figure 3.1.

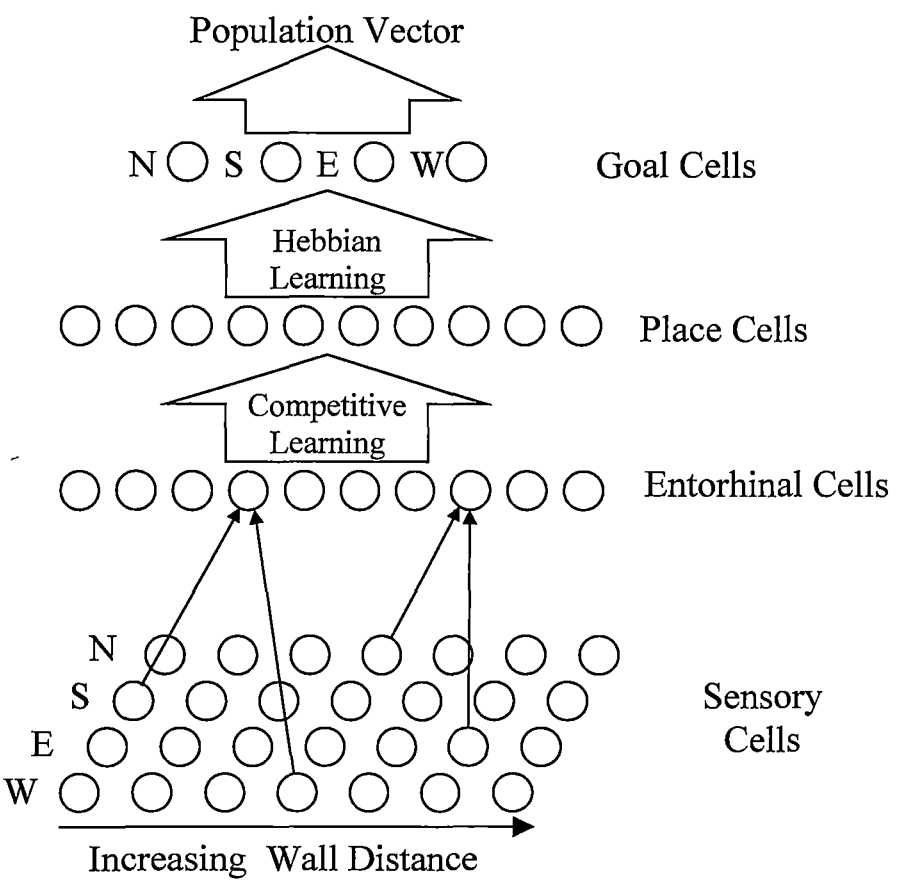

Figure 3.1: The place cell model of Burgess, Donnet and O'Keefe (1996; 1998).

The model was able to simulate many of the observed properties of place fields, including elongation of place fields near walls, and distortions of place fields when

the environment is similarly distorted. However in a more complex environment, the robot would be subject to perceptual aliasing problems. That is, in environments where the view from two distinct places may be identical or similar, place cells tuned to this view will not be able to distinguish between the two locales, and hence will have two place fields. This is not a desirable property if these place cells are to be used for navigation.

Place cells in the model of Gaussier and colleagues (2000; 2002) learn the expected bearings of visible landmarks when viewed from the corresponding environment location. The closer each landmark is to it's expected bearing, the higher the place cell activation. While this is potentially very useful, the resultant place fields do not resemble those of biological place cells. While the higher sensory resolution of the robot in this model would greatly reduce the risk of perceptual aliasing, it would nevertheless remain a problem with any view-only method.

3.1.3. Path Integration

Path integration alone is unable to produce a robust position estimate. Even if the system is perfectly accurate under normal circumstances, and this is almost impossible to achieve on a robotic platform, it is unable to provide initial localisation within the environment. Therefore, none of the models discussed in this chapter suggest a path integration only system for localisation. Instead, path integration is combined with the landmark or view cell information to overcome the perceptual aliasing problem.

Wan, Touretzky and Redish (Touretzky et al., 1994; Wan et al., 1994a, 1994b) combine the landmark information with current path estimate in a single step. The activation of place cells is determined using radial basis functions tuned to distances and bearings of landmarks, to the angles between landmarks, and to the path integrator coordinates. The expression for place cell activity is in the form of a product of Gaussians corresponding to each of these items. When any of this information is unavailable, the corresponding term drops out of the expression. This enables navigation in the dark and correct localisation when path integrator coordinates are known to be incorrect, such as when the robot enters the environment. For example, upon entering an environment, place cell activity is first calculated using the current view only. Each active place cell then recalls its learned position and orientation, and this information is used to reset the path integrator.

In contrast to Wan and colleagues (Touretzky et al., 1994; Wan et al., 1994a, 1994b), which is very abstract, Guazzelli, Bota and Arbib (2001) implement a more detailed

path integrator. The path integrator mimics the behaviour of the attractor model proposed by Kali and Dayan (2000). Path integration is implemented as a moving hill of activity on a two dimensional array of cells. The position of this hill represents the position of the animal, and is moved by applying movement information or information from the place cell layer. Connections between the path integration layer and the place cell system are mediated by feature detection layers as shown in Figure 3.2. Connections between these and other layers are modified using a form of competitive Hebbian learning. Each place cell responds to features present in the path integration layer and in the view layer.

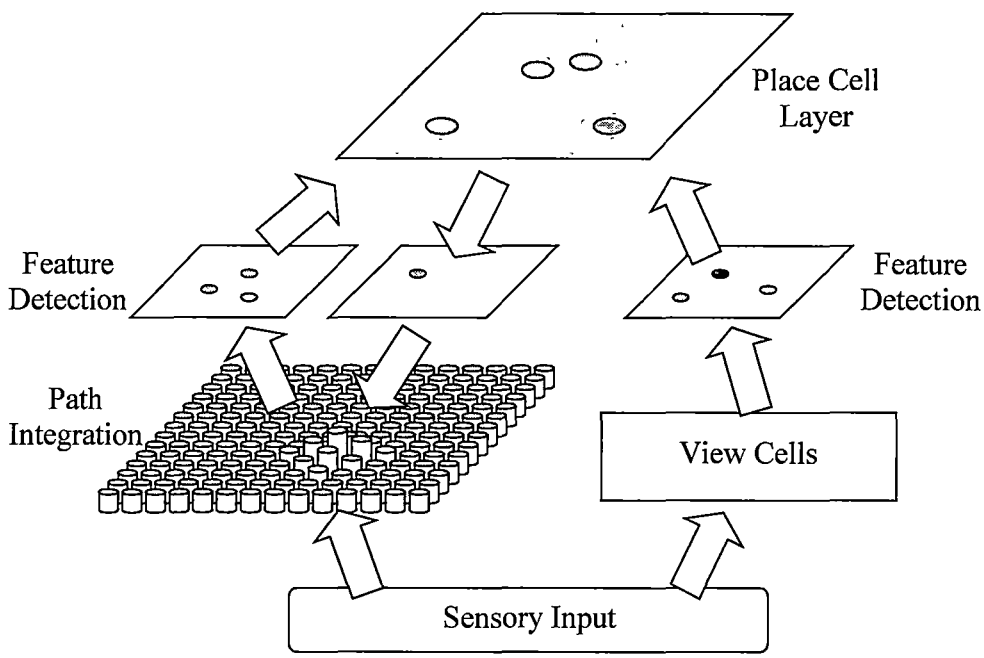

Figure 3.2: Simplified overview of the computational model of Guazelli, Bota and Arbib (2001).

Arleo and Gerstner (2000) use a similar path integration system, with the activation of path integration cells calculated as the Gaussian distance of the cell's preferred position from the estimated position. Place cells are activated by connections from the more complex view cells and path integration cells. These place cells are trained using Hebbian rules. The path integration system is recalibrated periodically using the population vector (Georgopoulos, Kettner, & Schwartz, 1988) of place field centres, which are set when a new place is first encountered. A similar system was also developed for maintaining the head direction estimate (Arleo & Gerstner, 2001), demonstrating the versatility of the approach.

summary

3.1.4. Kalman Filtering

All of the models discussed share a similar philosophy based on observations of biological systems. The basic process is to identify landmarks in the sensory view and extract information about the relative positions of those landmarks, this information can then be combined with estimates from a path integrator for more robust localisation. An alternative approach is to examine non-biological methods for achieving the same result and then to relate these back to the biological solutions.

A Kalman filter (Jazwinski, 1970) estimates the state of a dynamic system by combining a series of noisy state observations and on a model of how the state may change. In the case of robotic localisation, the state is the location of the robot, state observations are sensor input, and state changes are indicated by motor outputs, wheel rotation or some other measure of change in position. Under certain conditions, a Kalman filter can be shown to provide optimal update rules for combining uncertain information (Bousquet, Balakrishnan, & Honavar, 1997). Figure 3.3 depicts the basic Kalman filtering concept.

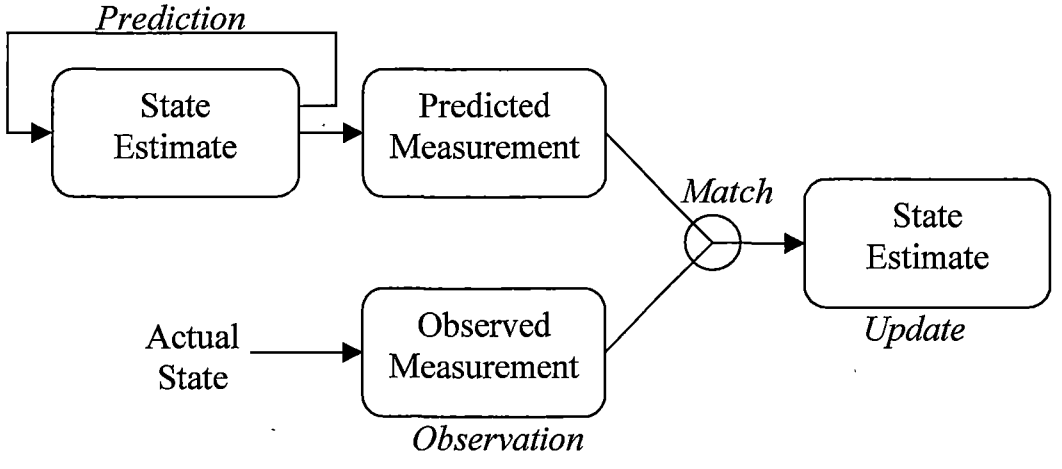

Figure 3.3: A schematic of Kalman filtering (adapted from Balakrishnan, Bhatt, & Honavar, 1998).

Kalman filtering has been used by many researchers (e.g. Dissanayake, Durrant-Whyte, & Bailey, 2000; Kleeman, 1992) for robot localisation, however few have related this back to the biological system. Balakrishnan and colleagues (Balakrishnan, Bousquet, & Honavar, 1999; Balakrishnan & Honavar, 1997; Bousquet et al., 1997) compare this with hippocampal localisation, as summarised in

Figure 3.4. They argue that the function of the hippocampus during localisation is the same as that of a Kalman filter.

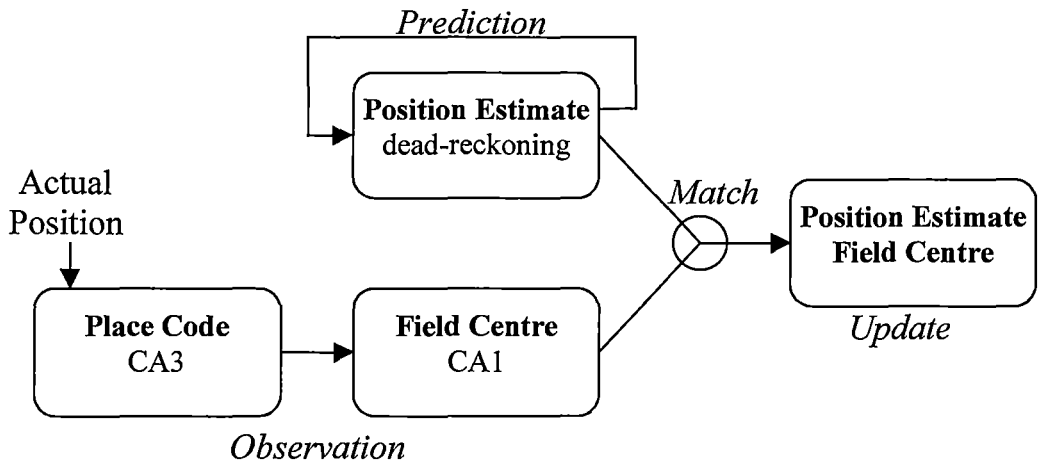

Figure 3.4: Hippocampal localisation and position update procedure (adapted from Bousquet et al., 1997).

From this observation, a computational model composed of five modules was developed as shown in Figure 3.5.

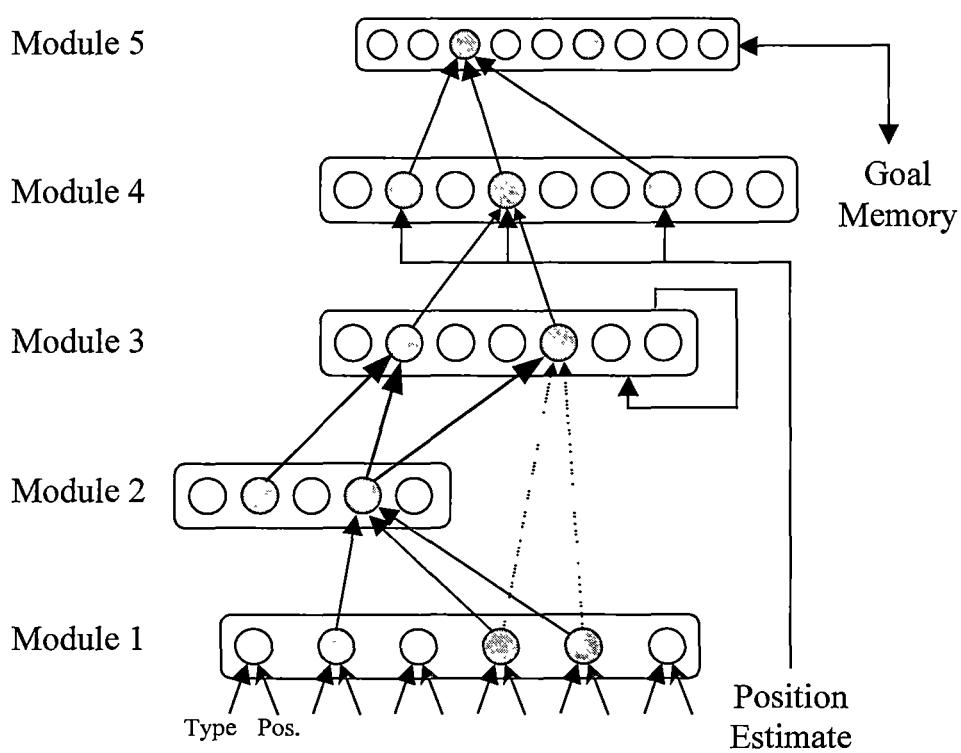

Figure 3.5: Hippocampal model of Balakrishnan and Colleagues (Balakrishnan et al., 1999; Balakrishnan & Honavar, 1997; Bousquet et al., 1997).

The computational model was designed to be a simplified simulation of modules 1 to 4. As is common with the biologically based model discussed in section 3.1.1, Module 1 view cell activation consists of a product of Gaussians tuned to the positions of perceived landmarks, with the type of landmark acting as an additional input. Module 2 cells respond to particular combinations of Module 1 cells, with new units being added if there is no Module 2 unit that matches the Module 1 activation. As each module 2 unit is added, it becomes associated with the current position estimate from the path integrator. The authors then use a modified form of the Kalman filtering algorithm to update the state estimate.

Lee and Recce (1997) also use a Kalman filter for localisation, but take a different, somewhat less biologically plausible, approach. The system was developed for a mobile robot called ARNE[9]. ARNE is constructed on a 300mm circular base with a two-wheel differential drive system. The robot is equipped with a single sonar sensor that is able to rotate, and is set to take readings at every 18 degrees. Sonar readings were used to build either a feature map[10] or an occupancy grid of the environment, and this map was used in conjunction with a Kalman filter to allow the robot to localise within the environment. While this system was not biologically based, it was later extended by Recce and Harris (Harris & Recce, 1997; Recce & Harris, 1996) to include some biologically inspired features.

A limitation of the original localisation system was that it was able to perform incremental localisation only. That is, given a starting position plus odometric information and sonar data, the robot was able to estimate the new location. The extensions of Recce and Harris also allow absolute localisation. That is, the ability to localise based on current sensory information only, as when the robot first enters the environment.

The extended system included an environment memory consisting of place cells. Each place cell stored a map representation in robot-centred coordinates. In each cycle, the partial map generated by the mapping system was presented to the place cells. Each place cell received a score based on the similarity of the stored map to the partial observed map. The retrieved maps of those place cells that fire strongly were then used to assist in localisation and mapping. The authors claim that this

---

[9] ARNE is another mid-sized mobile robot. See Lee (1996) for more information.

[10] A feature map simply stores the positions and orientations of labelled features.

place system is similar to Marr's (1971) auto-associative theory of hippocampal function, although this connection is not made clear.

3.2. Navigation

Place cell to goal cell connections are trained using 'one-shot' Hebbian learning as each goal is encountered. These goal cells form the basis for navigation, which will be discussed in section 3.2.

3.2.1. Coordinate Based Navigation

If the place cell model includes a metric path integration system, then navigation can be achieved using a simple coordinate based procedure. The robot remembers the path integrator coordinates of the goal location and compares these to the current position estimate. Vector subtraction of these coordinates gives the direction to the goal. Such a system was used by Touretzky, Wan and Redish (1994) in their simulations. Similarly, Balakrishnan, Bhatt and Honavar (1998) used this technique, however they also included a heuristic method for choosing an appropriate goal.

This simple coordinate based navigation cannot be applied without a metric estimate of the rodent position. While some of the models discussed in the previous section do not include this metric, Foster, Morris and Dayan (2000) developed a reinforcement learning algorithm for learning coordinates from place cell activation. The method was based on temporal difference learning (see Section 7.1). The system learned a 'value' function for each axis of the coordinate system. Value functions were updated using odometric input as a reward signal. The system was tested in a simulation of the Morris watermaze and the results showed good correspondence to the results for rodent experiments.

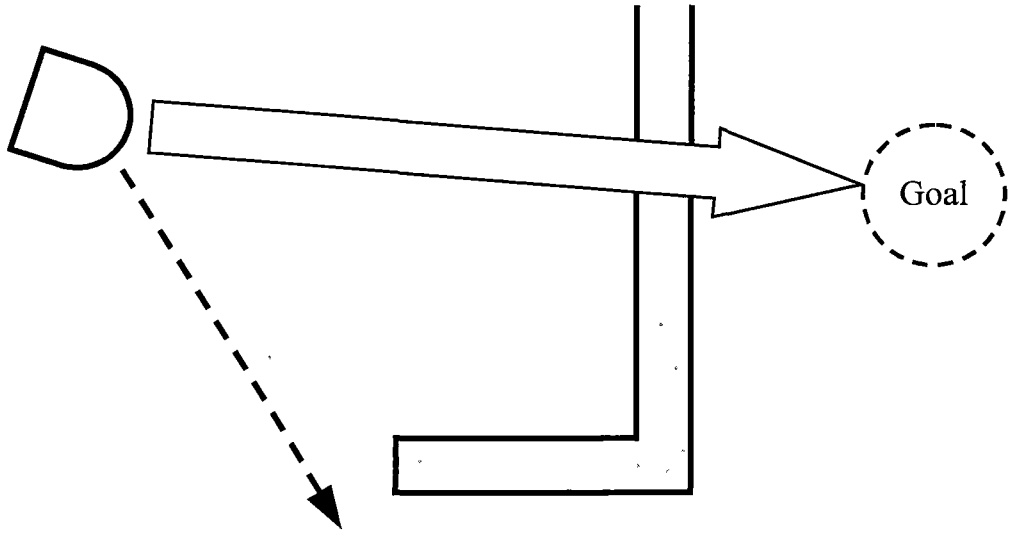

Figure 3.6: Coordinate-based navigation is unsuitable for complex environments. The large arrow shows the computed direction to the goal location, whereas the dashed arrow shows the optimal direction of movement.

The results of such experiments with robots and rodents are taken as evidence that rodents do maintain a coordinate representation of goal locations and the current position estimate. Unfortunately however, coordinate learning is not suitable for use in environments involving large or concave obstacles, as shown in Figure 3.6. Small convex obstacle can be navigated by moving along the object while also moving closer to the goal. However for larger and, in the worst case, concave obstacles (dead-ends) this technique will fail. Environments containing such obstacles will be referred to as *complex* environments.

3.2.2. Potential Fields

Gaussier and colleagues (2000; 2002) implemented navigation through the use of potential fields. When a goal is reached, the robot learns to associate nearby views with the goal by backing a small distance away from the goal and training view cells. This is repeated for movement in multiple directions. To return to the goal, the robot finds the view cell that best matches the current sensory input and moves in the direction indicated by that cell. Again, this method of navigation is only useful in simple environments, and will also be limited by the size of the environment. In addition, the complicated process of learning views for each goal limits the attractiveness of this approach.

Burgess, Donnet and O'Keefe (1996; 1998) developed a similar method that did not require a complicated learning procedure when the goal was reached. Associations

between goal cell and place cells were learned based on the direction of movement when the goal was reached, and the recency of place cell firing. While reducing the complexity of the learning procedure, this method does not solve the problem of navigating in large complex environments.

3.2.3. Reinforcement Learning

Reinforcement learning has long been used for low-level navigation, such as collision avoidance and wall following, and for navigation to a fixed goal (Sutton & Barto, 1998). Unfortunately, reinforcement learning algorithms perform poorly when navigating in environments with dynamic goal locations, such as watermaze tasks (Foster et al., 2000). Reinforcement learning algorithms learn the values associated with states and actions, with respect to the current goal. If the goal location is changed, the previously learned values interfere with the new task being learned. This problem will be discussed further in section 8.1.

Foster and colleagues (2000) developed a method for combining reinforcement and coordinate learning. The agent uses the actor-critic (see Section 7.1.1 for details) paradigm to choose between movement in each of eight discrete directions (as in conventional methods), as well as the direction computed by the coordinate system (see Section 3.2.1). In open environments, the critic will learn that the coordinate system may be trusted to compute an appropriate action, enabling efficient navigation with dynamic goals since the coordinates are goal independent. However in complex environments, the system will revert to the traditional reinforcement learning approach with the associated poor performance when goal locations change.

Arleo and Gerstner (2000; 2001) also used reinforcement learning for navigation. In particular, Watkins' Q-learning was used to learn a value function from a linear approximation based on place cell activity (see section 7.1 for details). In principle, a value function can be learned for each goal location, allowing navigation in both open and complex environments with dynamic goals. This technique does not make use of coordinate information, however it should be possible to combine the method with that of Foster and colleagues.

3.2.4. Hierarchical Navigation

The model of Trullier and Meyer (1997) consists of four layers corresponding to the entorhinal cortex, the dentate gyrus, area CA3 and goal cells. The entorhinal cortex cells learn orientation-dependent local views. The dentate gyrus is a form of short-term memory that remembers the current sequence of view cell firing as the animal

moves. The CA3 layer learns associations between place cells with neighbouring place fields. These associations are direction specific, so that a given connection may represent a neighbour to the North, for example.

Goal cells code for where the animal is in relation to each goal, with one goal cell for each direction (eg. North, East, South and West). When the animal reaches a goal, it triggers the CA3 connections in each direction and the propagation of neighbouring cells allows connections to be learned between the appropriate goal cell and all place cells in that direction. The major limitation of this form of navigation is that, like the coordinate techniques, the model is limited to simple environments without obstacles. This issue was addressed in a later refinement (Trullier & Meyer, 1998).

The extended model includes the notion of sub-goals. When the robot is at a location where goal information is not available, it moves around until it finds a location where goal information is available. At this point, a new set of sub-goal cells is recruited for the current location. Eventually enough sub-goal cells will be recruited to enable navigation from any location within a complex environment. However, this approach does not fit experimental observations, since it requires several visits to the goal location in order to learn enough sub-goals to enable successful navigation in complex environments. In contrast, rats are able to return to the goal after only one trial in the same situation.

Reinforcement learning has also been used with a similar hierarchy of goal states (Dayan & Hinton, 1993; Dietterich, 1998; Digney, 1996; Kaelbling, 1993a; Parr & Russell, 1997; Singh, 1992). These techniques show great promise for robust navigation in complex environments, and for reducing the time complexity of reinforcement learning algorithms (see Section 8.2.5).

3.3. Low-Level Navigation

While not a focus of this thesis, a complete system would also require complementary navigational systems for low-level tasks such as collision avoidance and exploration. Reinforcement learning is commonly used for this task and is the approach that will be taken for this thesis. Chapter 7 provides a review of temporal difference based reinforcement learning and demonstrates how that may be applied to the proposed system.

3.4. Summary

This chapter has reviewed some of the major biologically inspired systems for localisation, mapping and navigation. The general approach to localisation is quite consistent and involves the combination of view and odometric input to establish place units. However, the models differ in the way that this information is used, and in the way that cognitive maps are addressed. Some models maintain explicit representations of maps, whereas in other models, the maps are implicit or not present.

Many of the systems reviewed do not address the issue of navigation, and of those that do, the algorithms used are often restricted to small, open environments. Nevertheless, all of the algorithms reviewed have some merit. A combined approach involving reinforcement learning, coordinate systems and hierarchies of goals is most likely to provide an appropriate navigational solution.

Chapter 4. System Design

The main objective of this research is to develop an autonomous navigational system for a simulated mobile robot based on biological principles. The system will provide navigational abilities in typical real world environments, and should rely on simple sensory systems only.

Real world environments are typically complex and cluttered, with many obstacles, dead ends and potential shortcuts. They are also rarely static and may involve doors, movable obstructions, people or other robots. Ideally, a navigational system will be able to deal efficiently with all of these situations, without requiring complex and expensive sensors. Simple, inexpensive sensors that are commonly used on mobile robots include sonar and infrared rangefinders, bumpers for collision detection, and various devices for measuring odometric information, such as wheel rotation. A carefully designed bumper system is generally error and noise free. However, measurements from inexpensive rangefinders (especially sonar) and odometric devices may contain considerable noise, and/or be error prone. While generally noisy, sonar readings are also subject to misinterpretation resulting from specular reflections, echoes, and weak returns. Odometric readings are often very precise, but if measuring wheel rotations, for example, may introduce errors due to wheel slip and collisions, hence the resulting accuracy is usually quite poor, especially since these errors have a cumulative effect. The navigational system will need the ability to overcome the limitations of these sensors.

Two essential components of any navigational system are localisation and navigation. That is, the ability to determine the current position and the ability to deduce appropriate actions to reach the current goal. This chapter will describe the general design of the proposed system with reference to the previous models discussed in Chapter 3.

4.1. Localisation

Localisation can be divided into two parts; determining the current heading and determining the current position. Of these two, determining the current heading is considerably more critical as small errors in head direction can lead to large errors in the position estimate. Directional sensors (such as electronic magnetic compasses) are cheap and widely available, however such sensors are in general not very accurate. In particular, these may suffer from local disturbances due to the presence

of metal objects or power lines. Gyroscopes and accelerometers for tracking changes in direction are considerably more accurate, but these may introduce a small drift to the perceived heading, which is extremely undesirable when this reading is used to calculate changes in position. However, a careful combination of measurement devices can lead to reasonably robust head-direction systems (e.g. Benson, Stombaugh, Noguchi, Will, & Reid, 1998; Kim & Seong, 1996). Alternatively, an attractor based head-direction network, such as those used by Skaggs and colleagues (1995), may be used to maintain head-direction. As a further alternative, the place cell system preposed below could easily be modified to also correct head-direction. Given the many options available for maintaining a robust estimate of head-direction, this thesis will tackle only the more difficult problem of maintaining a positional estimate. However, care will be taken to ensure that the system is not overly dependent on an accurate head-direction estimate, although it is assumed that any global drift will be corrected.

Figure 4.1 shows the basic structure of the proposed system.

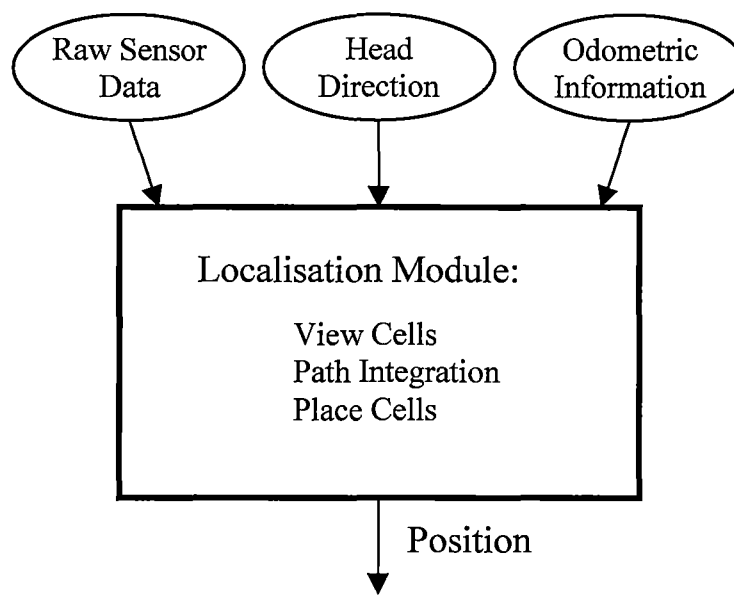

Figure 4.1: Localisation module.

The proposed localisation module is similar in structure to the models of Burgess, Donnet and O'Keefe (Burgess et al., 1996, 1998), and Arleo and Gerstner (2000). Sensor information is first used to establish the firing of a set of view cells. View cell output is then sent to place cells where the localisation is refined. Unlike the model of Burgess and colleagues, a path integration system is also included, in a similar way to the models of Wan, Touretzky and Redish (Touretzky et al., 1994;

Wan et al., 1994a, 1994b), Guazzelli, Bota and Arbib (2001), and Arleo and Gerstner (2000).

View cells in the proposed system will receive input from a set of range sensors. Unlike the models discussed in the previous chapter, view cells will learn an egocentric view of the environment. In contrast to the model of Burgess and colleagues (1996; 1998) view cells will be based purely on the features visible at the current robot heading. As with other models, view cell activation is a measure of the difference between the observed view and the cell's learned view.

As with the models of Arleo and Gerstner (2000), and Wan and colleagues (Touretzky et al., 1994; Wan et al., 1994a, 1994b), the proposed path integration system will be abstracted by simply storing the coordinates (in two dimensions) of the current position estimate. The influence of path integration input on place cell firing is calculated using the Gaussian distance between the stored position estimate and each place cell's stored field centre. The position estimate is primarily updated using odometric information. This system was chosen for ease of implementation and computational efficiency when compared to the model of Guazzelli and colleagues (2001), or the full attractor model proposed by Kali and Dayan (2000). The view cell and path integrator information is combined by place cells, where the necessary conversion is made from egocentric view cells to allocentric place cells. Finally, the population vector of place cells is used to correct the position estimate of the path integrator in a similar manner to other models.

The proposed navigational system will offer two major improvements over previous models. Firstly, view cells will use a new learning algorithm that is more suited to complex and unpredictable environments. Previous models have used a fixed Gaussian function tuned to environmental features. The new model will learn a similar function, but will adjust the centre, width and shape of the function online to provide better view discrimination without loss of generality. This new algorithm for learning view cells will be discussed in Chapter 5.

Secondly, Kudrimoti and colleagues (1995) provide evidence for the preconfiguration of the place cell system. To investigate the value of this preconfiguration, the path integrator coordinates of place cells will be fixed prior to training. This will allow the navigational system to make use of this known configuration for initialisation, resulting in improved exploration performance, and providing a mechanism for dead reckoning. The place cell system will be presented in Chapter 6.

4.2. Navigation

The navigational system can similarly be divided into two parts; low-level and high-level navigation. The low-level navigation system will provide the ability to deal with basic exploration and collision avoidance. This system should be relatively independent of the environment and will require some representation of the robot's current view. The high-level navigation system will be responsible for choosing the best direction of movement for reaching the current goal. This system will require as input the current goal and the current position. An overview of the navigation module is shown in Figure 4.2.

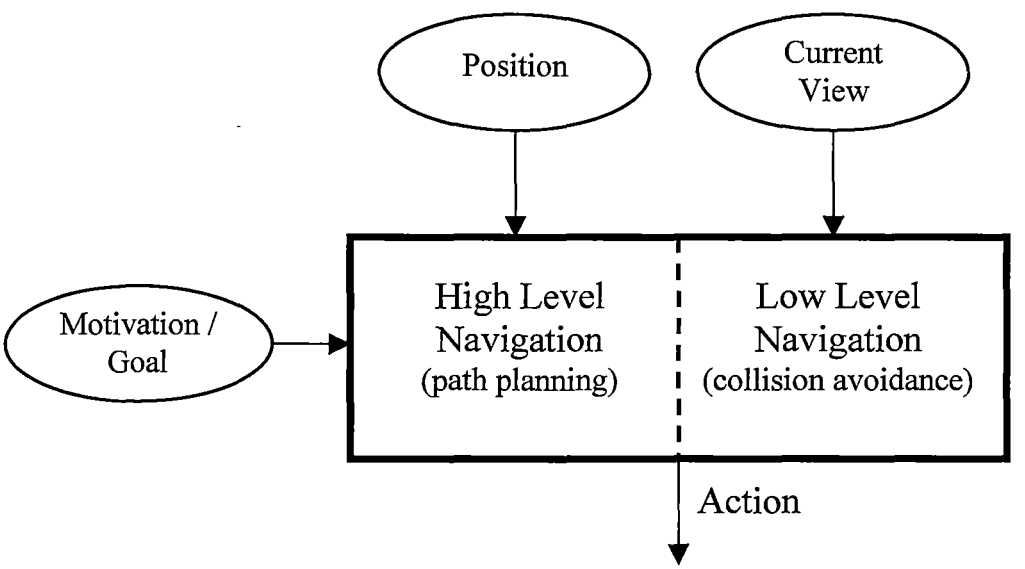

Figure 4.2: Navigation module

Reinforcement learning is commonly used to solve low-level navigational problems. The basal ganglia are widely regarded as a centre for reinforcement learning, and recent evidence suggests that temporal difference learning in particular, is an appropriate algorithm for modelling this system. Brain areas associated with processed sensory input send information to the basal ganglia; this information is an obvious minimal requirement for collision avoidance. Chapter 7 will briefly introduce the temporal difference learning algorithm and will examine the application of this algorithm to the low-level navigation task.

For high-level navigation, a system capable of dealing with complex and dynamic environments is required. Therefore, a coordinate-based system is not appropriate. The relationship between the hippocampal place cell system and reinforcement learning in the basal ganglia suggests that reinforcement learning may play a role in high-level navigation also. The high-level path planning system is therefore based

on the Q-learning approach of Arleo and Gerstner (2000). The major improvement over previous navigational systems, and the system of Arleo and Gerstner in particular, will be the development of a new reinforcement learning algorithm, called concurrent Q-learning (CQL), that provides complete goal independence. This will allow immediate navigation to any goal from any starting location. While similar to the DG-learning algorithm (Kaelbling, 1993b), CQL offers improved efficiency, especially in dynamic environments. A hierarchical form of CQL is also developed that greatly reduces the computational cost of the algorithm. The CQL algorithm is presented in Chapter 8.

4.3. Integration

The localisation and navigation algorithms required for the complete system are developed independently in the following chapters. This has the advantage of ensuring that, where possible, these algorithms retain applicability to other problems in the field of artificial intelligence. The disadvantage of this approach however, is that integration of these independent sub-systems is more difficult. Issues arising from this integration are discussed in Chapter 9.

Although the current goal is given to the high-level navigation system, a mechanism is needed to convert that goal into the place representation. As with the models of Burgess and colleagues (Burgess et al., 1996, 1998), and Balakrishnan and colleagues (Balakrishnan et al., 1999; Balakrishnan & Honavar, 1997; Bousquet et al., 1997), a goal system is developed in Section 9.1.1 to learn associations between goal locations and place cells.

Section 9.1.2 deals with the integration of the place cell system and the high-level navigation system. Section 9.1.3 discusses the method used for combining the output of the low-level and high-level navigation systems. Initialisation of the high-level navigation system, using information available due to the preconfiguration of the place cell system, is discussed in Section 9.2. This section also discusses other initialisation and pre-training issues. Finally, Section 9.3 discusses the performance of the complete integrated system.

Chapter 5. View Cell System

View cells should be able to accurately capture the salient information of the view at a particular location and orientation. While substantial changes in that position and orientation should result in a significantly decreased firing of the view cell, minor changes should not result in a major change. Many researchers have found a simple Gaussian function sufficient to model view cells. However, these experiments take place in simple environments (eg. rectangular or cylindrical) and/or it is assumed that the view cell input has already been significantly processed (eg. by finding the orthogonal distance to walls).

A view cell in more complex environments with no prior processing of sensory information would need to be more robust. Figure 5.1 shows a simple robot with two range sensors. Small changes in the position and orientation of the robot will not result in a significant change in the first sensor reading. However, a small change may result in a significantly different reading from the second sensor, due to the acute angle of the incident wall. In addition, the proximity of the corner means that the range of distances perceived by the second sensor will have an abrupt lower bound.

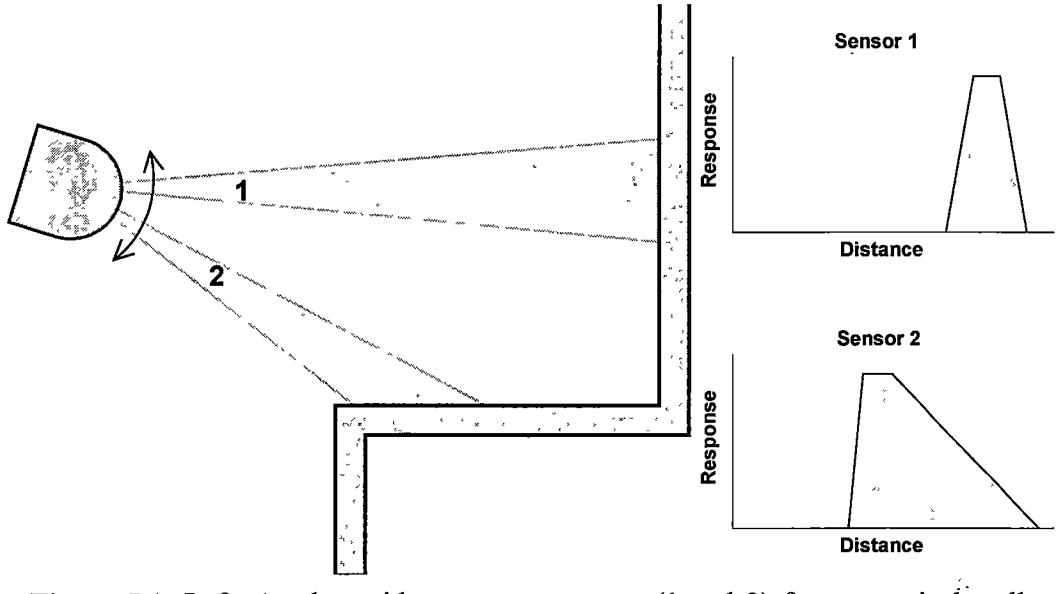

Figure 5.1: Left: A robot with two range sensors (1 and 2) faces a typical wall section. Right: Response functions that would capture this view, while allowing small variances in robot position and orientation.

A cell to capture this view would need to have different response functions for each sensor in order to maintain a robust output across small variations in position and

heading, while retaining the ability to differentiate conceptually different views. Furthermore, a robust system will need a method whereby these response functions may be learned on-line.

Section 5.1 presents a general neural model with properties that are suitable for learning view cells in this context. Section 5.2 discusses the application of this model to the current problem.

5.1. Adaptive Response Function Neurons

Biological neurons are typically modelled using a linear response function (with a sigmoidal activation function). The electrical potential of a neuron is calculated as the weighted sum of its inputs, with the weights representing the synaptic efficiencies of the input connections. While the biological system is considerably more complex than this, it can be assumed that the response functions of the majority of neurons are at least monotonic. Biological neurons do exist, however, that respond in a selective way to input. This may be due to either the physical properties of some sensory neurons, or to the topology of the network containing the locally responsive neuron (Moody & Darken, 1989)

Certain classes of artificial neural network also contain neurons that are locally responsive to certain input levels. These include self-organising maps (SOMs) (Kohonen, 1995) and radial basis function (RBF) networks (Moody & Darken, 1989). The output of a SOM neuron is typically a distance measure from the supplied input to a stored exemplar, while the response function of RBF neurons is typically a Gaussian.

The outputs of SOM neurons are compared to find a winning neuron, hence the critical parameters for a SOM are just the stored exemplars - that is, the centres of the response functions. The locations of the centres of the response functions are randomly initialised. For each example presentation the winning neuron (that neuron whose exemplar is closest to the input), and some surrounding "neighbourhood" of neurons, have their exemplars shifted towards the given input pattern.

The outputs of RBF neurons, on the other hand, are often passed onto a second layer of neurons, which are typically trained using a supervised gradient descent rule. Thus, both the centres and widths of the response functions of RBF neurons are important. Many techniques have been proposed for determining appropriate centres and widths of the basis functions of RBF networks.

One solution is to find RBF centres by applying a clustering algorithm such as K-means and determining appropriate widths using techniques such as "P-nearest neighbours" (Bruzzone & Prieto, 1999; Moody & Darken, 1989). In order to produce more compact RBF networks, Leonardis and Bischof (1998) propose a method of pruning based on the minimum description length (MDL) principle. None of these training methods can be employed on-line.

The resource-allocating network (RAN) of Platt (1991) adds neurons if the network error is high and adjusts the centres of existing neurons if the error is low. The width of the Gaussian response functions is reduced as new neurons are added. While the RAN can be trained on-line, it has the disadvantage of having an indeterminate network size.

By considering a biologically plausible sub-network for the formation of locally-tuned neurons, a training method was developed that can be used on-line. The training algorithm independently adjusts the centres, widths and shapes of locally tuned response functions for each input to the neuron.

5.1.1. The Neural Model

Within the field of artificial neural networks (ANNs), a frequency model of biological neurons is commonly used. The output of such a neuron represents the firing frequency of the neuron. The activation function is typically a sigmoid, and the input response is usually linear with individually adjustable weights representing synaptic efficiencies. This model will be used to develop the adaptive response function neuron (ARFN).

Consider a cortical neuron that receives input from both an excitatory and an inhibitory interneuron. Now suppose that each of these interneurons is excited by a common cortical input (see Figure 5.2). With appropriate choices for thresholds, the output neuron, which we shall call the ARFN, will have a Gaussian like response to the cortical input. Equation 5.1 gives the input response function, $R(x)$, for the ARFN.

$$R(x) = \frac{s_e}{1 + e^{t_e - g_e x}} - \frac{s_i}{1 + e^{t_i - g_i x}} \qquad \textbf{5.1}$$

where s_e and s_i are the synaptic efficiencies of the interneuron→ARFN connections for the excitatory and inhibitory interneurons respectively; g_e and g_i are the synaptic

efficiencies of the input→interneuron connections; and t_e and t_i are the synaptic efficiencies of the threshold→interneuron connections.

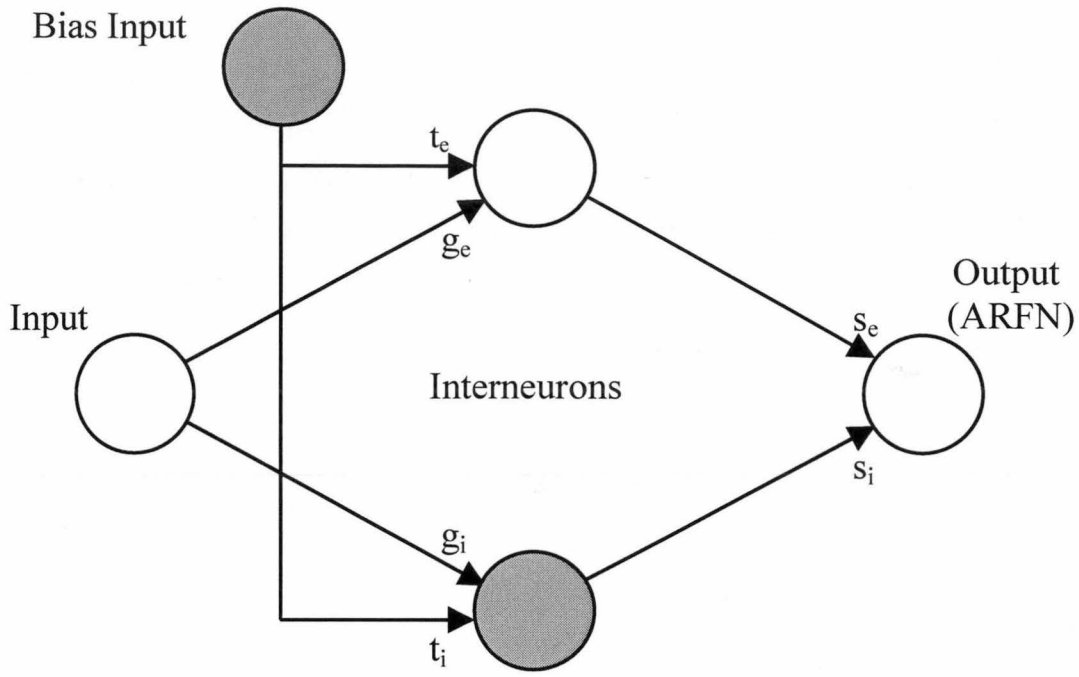

Figure 5.2: A neuron arrangement to implement a guassian-like response function. White neurons are excitatory, grey neurons are inhibitory. t_e, t_i, g_e, g_i, s_e and s_i are the synaptic efficiencies of the indicated connections.

In Figure 5.2 we see that there are six synaptic connections that could be modified. Two of these (g_i and g_e) are from the input to the two interneurons. Modifying the synaptic efficiencies of these neurons would effectively modify the slopes (gain) of the corresponding sigmoid activation functions of the interneurons. These could potentially be modified independently to create an asymmetrical output response function.

Another two synapses (t_e and t_i) occur between the inhibitory bias input and the interneurons. Modifying the synaptic efficiencies of these neurons would alter the threshold of the two sigmoids. This would adjust the centre and width of the output response function.

Finally the synaptic connections (s_e and s_i) between the interneurons and the ARFN could be modified. It is not clear that modification of these synapses would perform any useful function. Therefore these synapses have been ignored (set to a value of 1.0) in the development of the ARFN, and will be omitted in the following discussion.

Figure 5.2 shows the network topology for a single input ARFN. For an ARFN with multiple inputs, each input has its own pair of interneurons, which allow independent response functions to develop. The inhibitory bias input is shared by all interneurons as shown in Figure 5.3.

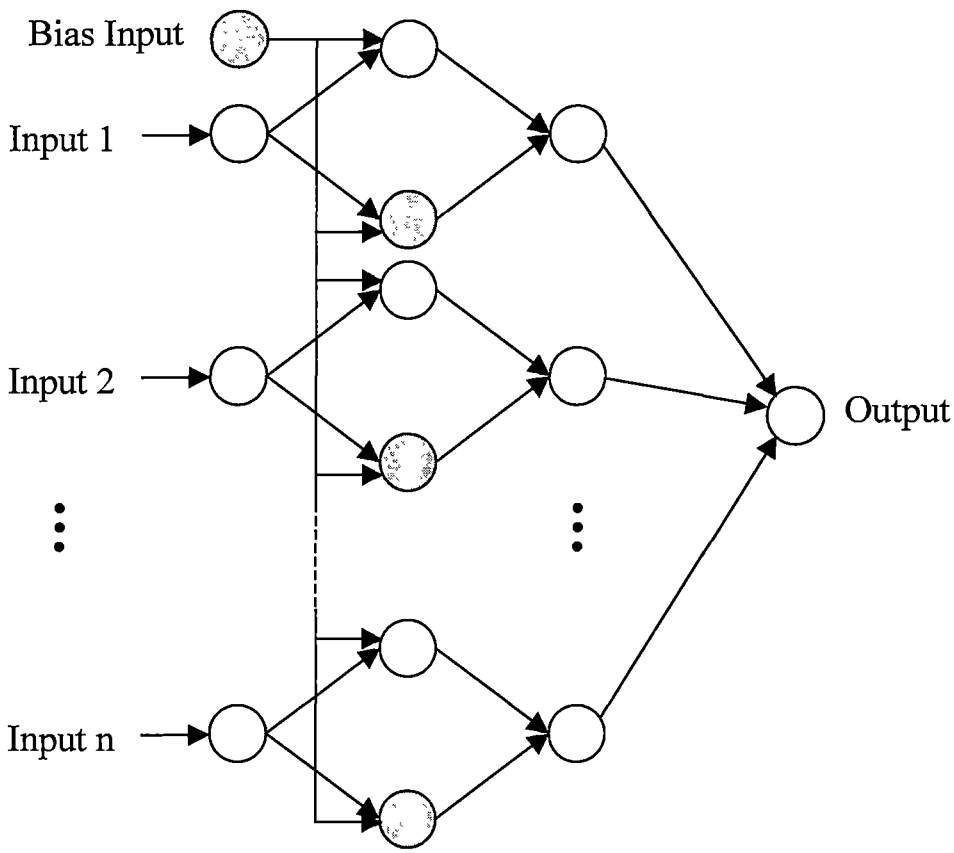

Figure 5.3: Synaptic connections for a multiple-input ARFN.

In Figure 5.3, the response for each input is combined to form a single output. The most appropriate method for combining the individual response functions will depend upon the application. One method would be to treat the output as a standard neuron and calculate the output using a sigmoid activation function, adjusting the threshold and gain of this function to suit the level of discrimination required. Alternatively, the output could simply be the average of each individual response function. This approach was taken for the remainder of this section. If greater sensitivity is required, the output can be the product of the individual response functions, and this method was used for view cells in section 5.2.

5.1.2. Training

Training the thresholds of the interneurons is straightforward. In the case of the excitatory neuron, if the response of the interneuron is high, the threshold should be

trained up, if it is low it should be trained down. The opposite should occur for the inhibitory interneuron. These learning rules are shown in equations 5.2 and 5.3 below.

$$\Delta t_e = \eta_t (r_e - \alpha) \qquad \textbf{5.2}$$

$$\Delta t_i = \eta_t [\alpha - (1 - r_i)] \qquad \textbf{5.3}$$

where η_t is the training rate for thresholds; r_e and r_i are the outputs of the excitatory and inhibitory interneurons respectively; and α is a parameter determining the equilibrium position for the training rule.

In terms of a possible biological implementation, it is assumed that if the neuron is to be trained, the bias input is set high; and if the neuron is not to be trained, the bias input is set low. This could be achieved by feedback connections after some form of competition has determined those neurons to be trained. Thus equations 5.2 and 5.3 can be considered modified Hebbian learning rules.

If the parameter α in equations 5.2 and 5.3 above is greater than 0.5, then as well as adjusting the centre of the response function, the width will also be adjusted in an intuitive way. As shown in Figure 5.4, if the output of both the excitatory and inhibitory interneuron is either high or low, the response function will expand as well as moving the centre of the response function towards the input value. If the output of the excitatory interneuron is high and the output of the inhibitory neuron is low, the response function will contract towards the input. If the network is consistently trained on a small range of inputs, the width of the response function will be small; whereas if the input range is wide, the width of the response function will be large.

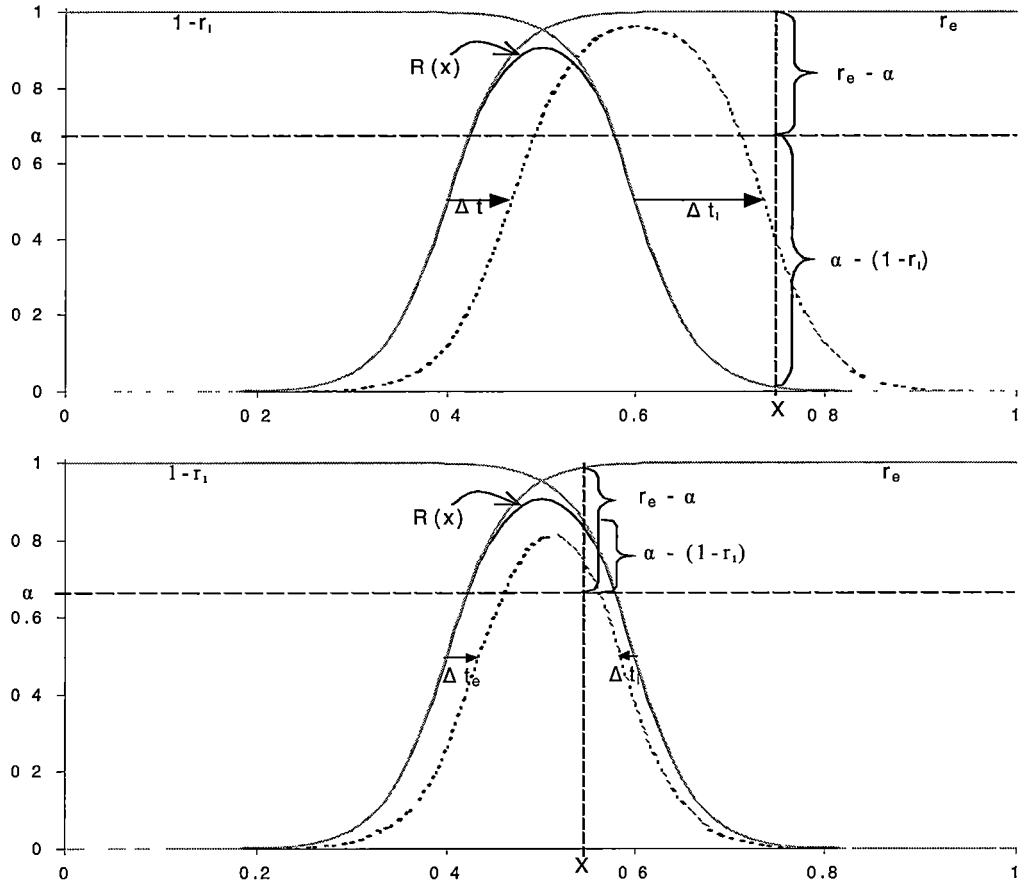

Figure 5.4: The effect of threshold training on the width and centre of the response function, $R(x)$, for $\alpha > 0.5$ and input X. r_e is the output of the excitatory interneuron, $1-r_i$ is the inverse of the output of the inhibitory interneuron. (a) For the input X shown, $\alpha-(1-r_i) > r_e-\alpha > 0$, hence the thresholds of the interneurons both increase resulting in the expansion of $R(x)$ (dotted line). (b) For the input X shown, $\alpha-(1-r_i) < 0 < r_e-\alpha$, hence the thresholds of the interneurons move towards the input resulting in the contraction of $R(x)$ (dotted line).

Unfortunately, as the width of the response function decreases, the amplitude will also reduce until the excitatory and inhibitory sigmoids completely cancel each other. Conversely, the amplitude of the response function increases as the width increases. To avoid this, the gains of the sigmoids must increase as the width of the response function decreases.

If the output of the excitatory interneuron is less than 0.5, decreasing the gain of the sigmoid (decreasing the synaptic efficiency of the input→interneuron connection) will increase the response of the neuron to that input value. Similarly, if the output is greater than 0.5, increasing the gain of the sigmoid will increase the response of the neuron to that input value. Since we want the output of the neuron to increase for a

particular input value after training, we could devise learning rules for the input→interneuron synapses as shown in equations 5.4 and 5.5 below.

$$\Delta g_e = \eta_g \left(r_e - 0.5 \right) \qquad \textbf{5.4}$$

$$\Delta g_i = \eta_g \left[\left(1 - r_i \right) - 0.5 \right] \qquad \textbf{5.5}$$

where η_g is the training rate for gain; and g_e and g_i are the synaptic weights of the gain connections for the excitatory and inhibitory interneurons respectively.

Unfortunately, these intuitive rules do not produce desirable behaviour. If these rules are used, the excitatory interneuron places too much importance on outlying high inputs, and vice versa for the inhibitory interneuron. The modified rules in equations 5.6 and 5.7 overcome this problem.

$$\Delta g_e = \eta_g \left(r_e - 0.5 \right) \left(\beta - r_e \right) \qquad \textbf{5.6}$$

$$\Delta g_i = \eta_g \left[\left(1 - r_i \right) - 0.5 \right] \left[\beta - \left(1 - r_i \right) \right] \qquad \textbf{5.7}$$

where $\beta > \alpha > 0.5$. The β-term modifies the effect of outlying high inputs (or low inputs for the inhibitory neuron) and has the effect of decreasing the gain or slope of the sigmoid for inputs in this region. This rule has the undesirable effect that it is now impossible to learn flat-topped response functions. While it may be possible to develop training rules that overcome this problem, in practice it has been found that these rules produce a very good fit to the presented data as can be seen in Ollington and Vamplew (2003). However, these rules require a number of parameters to be tuned and the effect of tuning is not always intuitive. Therefore, results for the remainder of this section were obtained using the much simpler and more robust rules below.

We can define a sigmoid by two points, so to define the ARFN response function we need to find four points. We choose the x-coords to be x_{p1}, x_{p2}, x_{p3}, and x_{p4} representing the $p1^{th}$, $p2^{th}$, $p3^{th}$ and $p4^{th}$ percentiles of the dataset respectively (normally $p1 = 1\text{-}p4$, and $p2 = 1\text{-}p3$). These points can easily be calculated offline, or found online using the simple learning rule:

$$\Delta x_p = \begin{cases} \eta(1-p) & \text{if } x_i < x_p \\ \eta p & \text{otherwise} \end{cases} \qquad \textbf{5.8}$$

where η is the training rate and x_t is the current training data.

Figure 5.5 shows how y-coords y_1 and y_2 may be chosen to complete the definition of the response function.

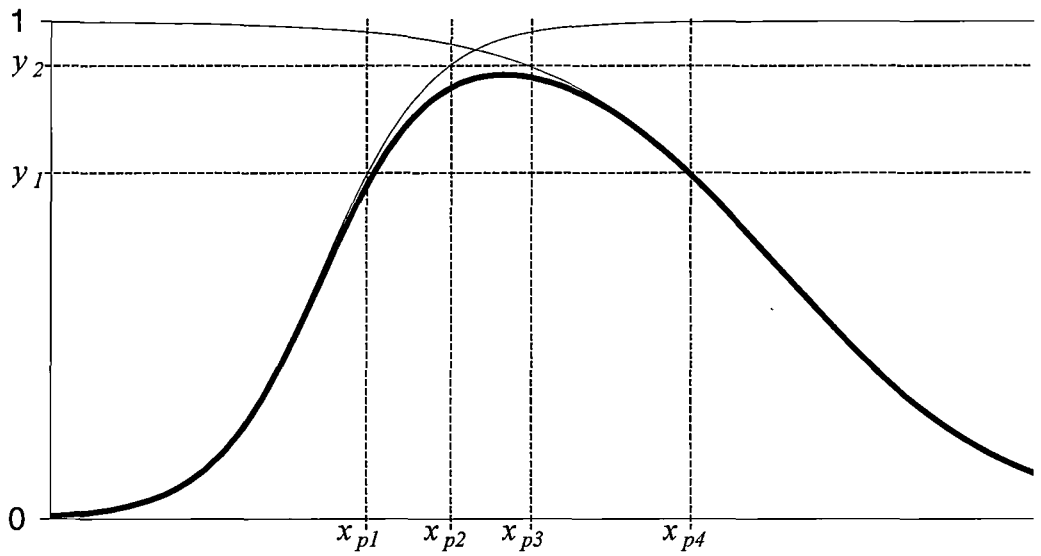

Figure 5.5: Demonstrates how the four percentile points, x_{p1}, x_{p2}, x_{p3}, and x_{p4}, define the response function.

It would obviously be useful to find percentile points near the periphery of the dataset, however in practice these points can be difficult to estimate. This is because examples falling outside these points will occur very rarely or may not occur at all during training. If points that are more central are used, it may be necessary to choose high values for y_1 and y_2 in order to maintain good generalisation. In most cases, the following parameters have been found to be effective: $p1$ = 10%, $p2$ = 25%, $p3$ = 75% and $p4$ = 90%; and y_1 = 0.95 and y_2 = 0.98. Parameter choice will be discussed further below.

5.1.3. Validating the Model: Classification

The primary motivation for the development was for view classification as described at the beginning of the chapter. Therefore, a synthetic classification dataset was devised to test the suitability of ARFNs for this purpose. The dataset represents three distinct "views" that may confront a mobile robot with two range sensors as shown in Figure 5.6.

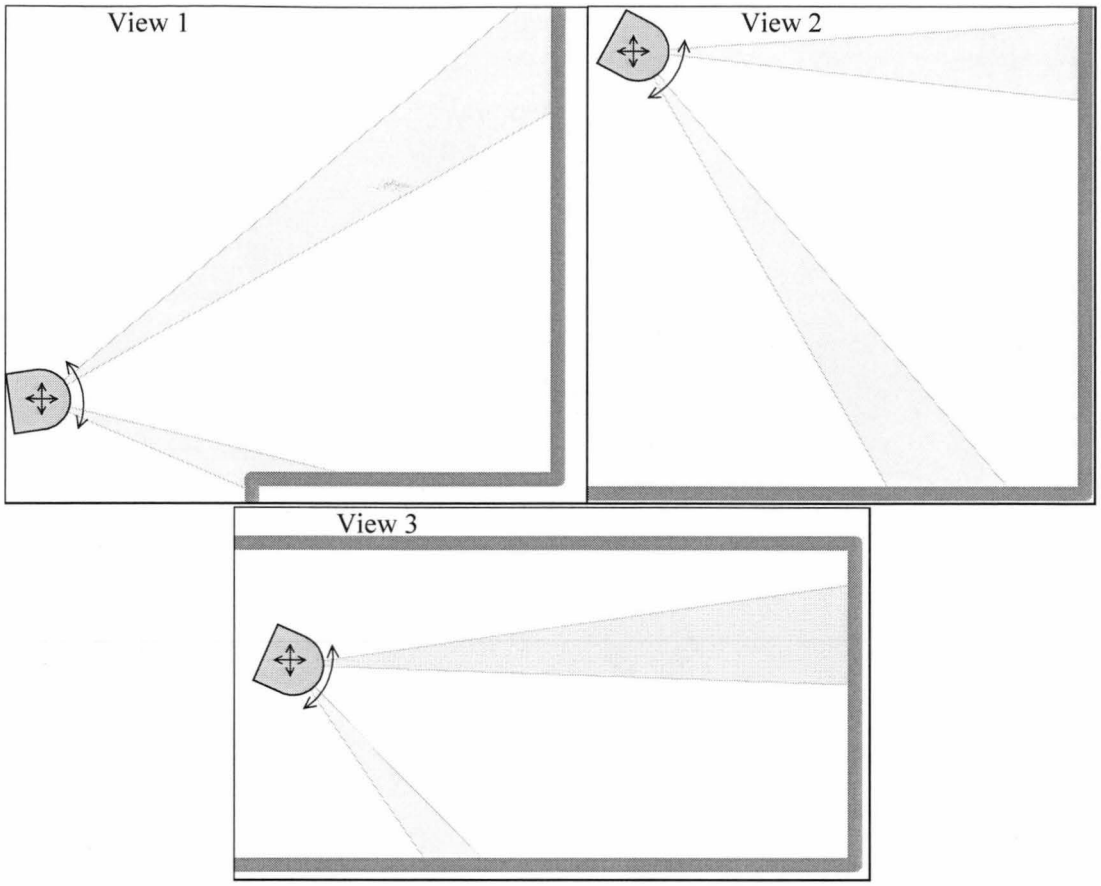

Figure 5.6: Three robot "views" used for generating the view dataset.

The robot has two range sensors, one angled 30 degrees to the robot's left, and one angled 30 degrees to the robot's right. The dataset consists of ranges that would be recorded for these sensors assuming complete accuracy. However, the robot's position and orientation was slightly different for each example in the dataset. 100 examples were generated for each view, the complete dataset can be found in Appendix C.

The ARFN was compared to a backpropagation network, and to a network trained using a fixed-width Gaussian where the width of the Gaussian was a tunable parameter. All networks were trained using 50% of the dataset and tested using the other 50%. The ARFN network achieved a mean accuracy of 99.6±0.1% on the training data, and 98.9±0.2% on the test data. The backpropagation network achieved a mean accuracy of 99.3±0.1% on the training data, and 99.2±0.2% on the test data. The Gaussian network achieved a mean accuracy of 95.2±0.4% on the training data, and 94.5±0.5% on the test data. The results show that there exist even simple datasets for which a fixed-width Gaussian is not the best choice, and where even a simple supervised ARFN network achieves similar performance to the much slower learning backprop network.

To help visualise the response functions, the ARFN was also trained on the entire data set and the resultant response functions for a single trial were plotted along with the frequency distribution of the input data for each category. The response functions are shown in Figure 5.7.

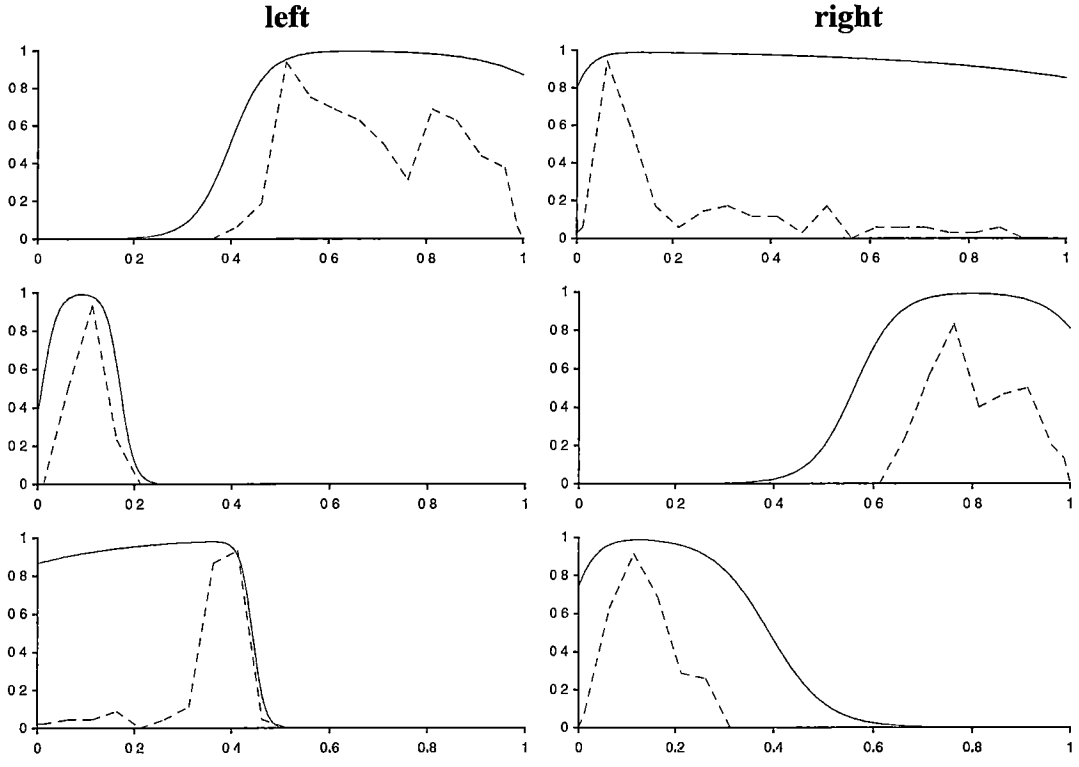

Figure 5.7: Response functions learned for view 1, 2 and 3 (top to bottom) for the left and right sensors when training on the view dataset. The solid line show the response function learned, and the dotted line shows the frequency distribution (not to scale) of the dataset.

Figure 5.7 clearly demonstrates the potential of ARFNs. A narrow response function would easily distinguish between view 2 and 3 based on the value for the right sensor. However, there would be difficulty distinguishing between view 1 and 2, since high values for the right sensor would not be recognised as potentially belonging to view 1. Conversely, a wide response function would not be able to distinguish between view 1 and 3, since the functions for the left sensor would overlap significantly. ARFNs are able to learn the sharp distinction between view 1 and 3 readings for the left sensor, while also learning the broad function required for view 1's right sensor.

The ARFN was also tested on two real world data sets obtained from the UCI repository of machine learning databases (Newman, Hettich, Blake, & Merz, 1998), namely the Iris dataset of Fisher (1936) and the Satellite dataset, which will be

referred to as the *View* and *Sat* datasets. The results for all three datasets are summarised in Table 5.1.

Table 5.1: Comparison of mean accuracy of supervised backpropagation, ARFN and Gaussian networks. Results are for 100 independent trials with 50% of the dataset chosen for training and the other 50% used for testing. 95% confidence intervals are also shown. The backpropagation networks had 2, 3, and 3 hidden nodes for the *View*, *Iris* and *Sat* datasets respectively. The Gaussian width was 1.5, 0.6 and 0.8 for the *View*, *Iris* and *Sat* datasets respectively.

	View Dataset		*Iris* Dataset		*Sat* Dataset	
	Training	Test	Training	Test	Training	Test
Backprop	99.3±0.1	99.2±0.2	99.0±0.2	95.0±0.4	79.3±0.2	78.9±0.2
Gaussian	95.2±0.4	94.5±0.5	93.3±0.4	91.1±0.7	74.9±0.1	74.5±0.1
ARFN	99.6±0.1	98.9±0.2	95.8±0.4	93.5±0.5	67.8±0.2	67.7±0.2
ARFN†					74.4±0.2†	74.1±0.3†

† sat results for alternative parameter choice

Note that for the *Sat* dataset, the parameters suggested in the previous section do not give good results compared to the tuned Gaussian network. Since it is difficult to correctly classify more than 80% of this dataset, it is not useful to search for percentile points that are beyond or close to this range. When the network was retested with $p1 = 25\%$, $p2 = 40\%$, $p3 = 60\%$ and $p4 = 75\%$, significantly better results were observed. However, neither the Gaussian nor the ARFN networks were able to achieve results comparable to the backprop network for this dataset.

Multi-layer networks

To improve performance on more difficult data, the number of ARFNs must be increased. This was achieved by adding an extra layer to the network with the hidden layer consisting of ARFNs, and the consisting of regular neurons. The hidden layer was trained using an unsupervised learning scheme, while the output layer was trained using supervised learning.

The network was again compared to a similar network of Gaussian neurons, with the hidden layer of this network trained as a Kohonen-style Self-Organising Map. Hidden neurons were arranged into a 2D square map for this purpose. This training method, however, proved to be unsuitable for training ARFNs since the network was very sensitive to the neighbourhood size chosen for training. Instead, the hidden ARFN layer was initialised so that all neurons produced a high response to all of the

data (ie x_{p1} = -0.1, x_{p2} = 0.0, x_{p3} = 1.0, and x_{p4} = 1.1) and, for each example presentation, only the winning neuron was trained (with ties broken randomly). As neurons become more specialised, other neurons that have not previously been trained are found to provide the best match for new data. In this way, all neurons quickly settle into a particular niche of the dataspace.

Figure 5.8 shows performance of the Gaussian and ARFN networks for different numbers of hidden neurons.

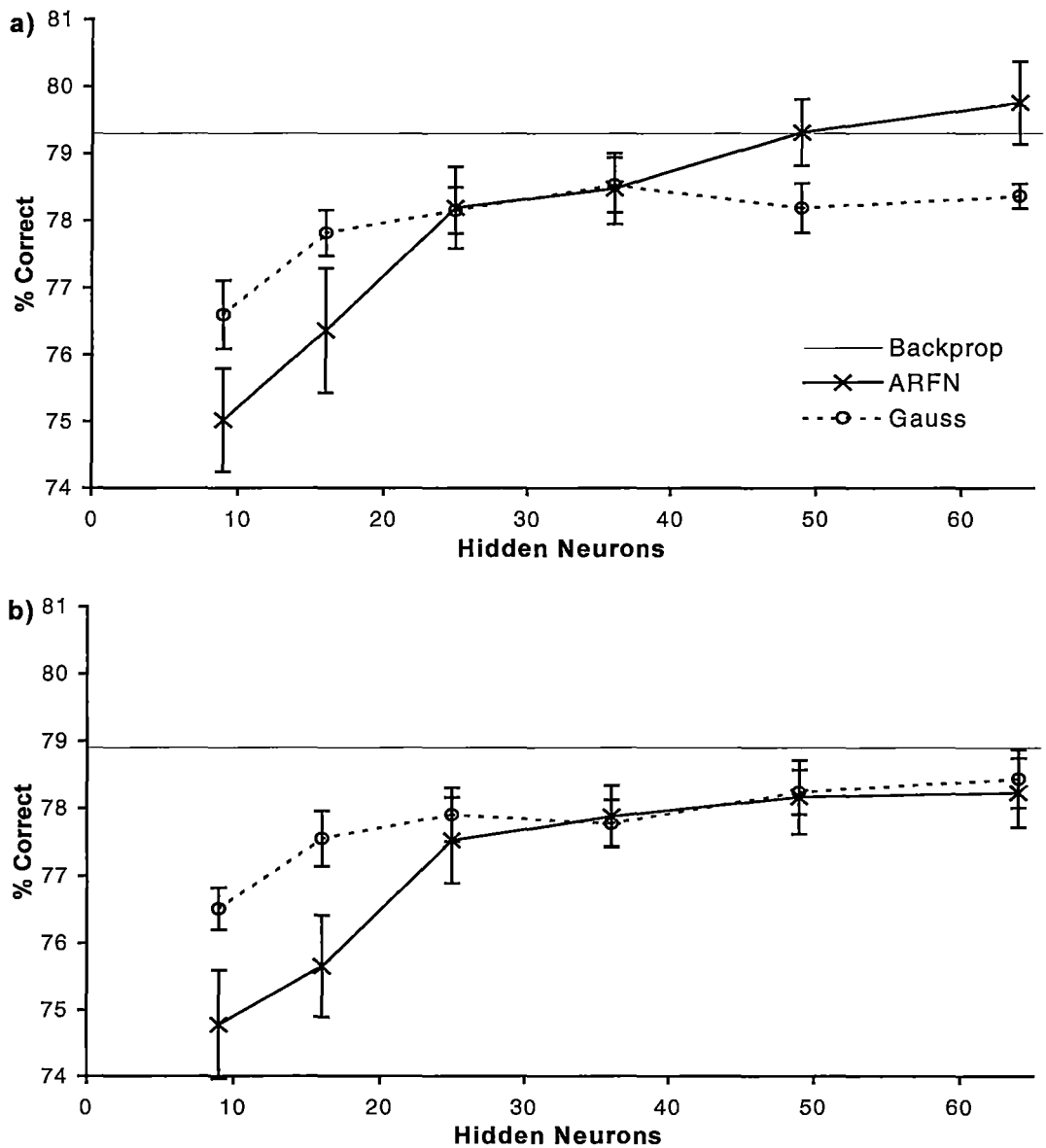

Figure 5.8: The performance of ARFNs and fixed-width Gaussian neurons on the *Sat* data set. Results for the training (a) and test (b) data sets are shown, error bars indicate 95% confidence intervals. The backprop line is for reference only and shows the mean performance for the optimal number of hidden nodes.

Both the ARFN and Gaussian networks show similar performance on the unseen test data, however both fall slightly short of the backprop network. The ARFN network does not perform well with fewer hidden nodes, but with 64 hidden nodes performs considerably better on the training data that the Gaussian network, and slightly better than the backprop network. This is probably partly due to the unsupervised training scheme used. The method enables all neurons to quickly find a niche within the dataset, achieving maximum separability while minimising the number of useless neurons.

5.1.4. Summary

The adaptive response function neuron presented is able to achieve a better fit to the presented data than a neuron using a fixed-width Gaussian response function. ARFNs trained in a supervised manner are able to perform better than fixed-width Gaussians on some datasets, and equivalently on others. Unlike many methods for adjusting response function widths, ARFNs may be continually updated online, and may learn asymmetrically shaped response functions. It appears that these properties make ARFNs particularly well suited to the types of data that are likely to be presented to the proposed view cell system.

While backpropagation algorithm performs as well or better on the datasets tested, this algorithm is not well suited to fast on-line learning. In addition, backprop is not biologically plausible and does not fit well with the biological data for view cells.

Aside from practical applications, ARFNs also provide some biological justification for other networks using local response functions. The ARFN is not a model of any particular biological system. However, it is certainly possible, given the neuron types and numbers available, that such neurons could exist in the neocortex or archicortex. Using only simple Hebbian-like training rules, ARFNs are able to adapt the width, shape and centres of locally tuned response functions. In addition, an even simpler training scheme can be used to provide similar results, while being less dependant on parameter choice.

5.2. ARFNs as View Cells

To test the viability of ARFNs as view cells, a simulated robot (see Appendix A) undergoing a collision avoidance task was used to generate training data. The robot had 9 range sensors at angles of –135, -90, -45, -22.5, 0, 22.5, 45, 90, and 135 degrees with respect to the orientation of the robot. The environment consisted of a simple maze, as shown in Figure 5.13.

The ARFNs were trained using the on-line training rule given in equation 5.8, and the unsupervised learning scheme described in section 5.1.3. To improve long-term stability, the learning rate for each view cell was slightly reduced (by 0.1%) each time that cell was trained.

Figure 5.9 shows the views learned by 16 of these cells.

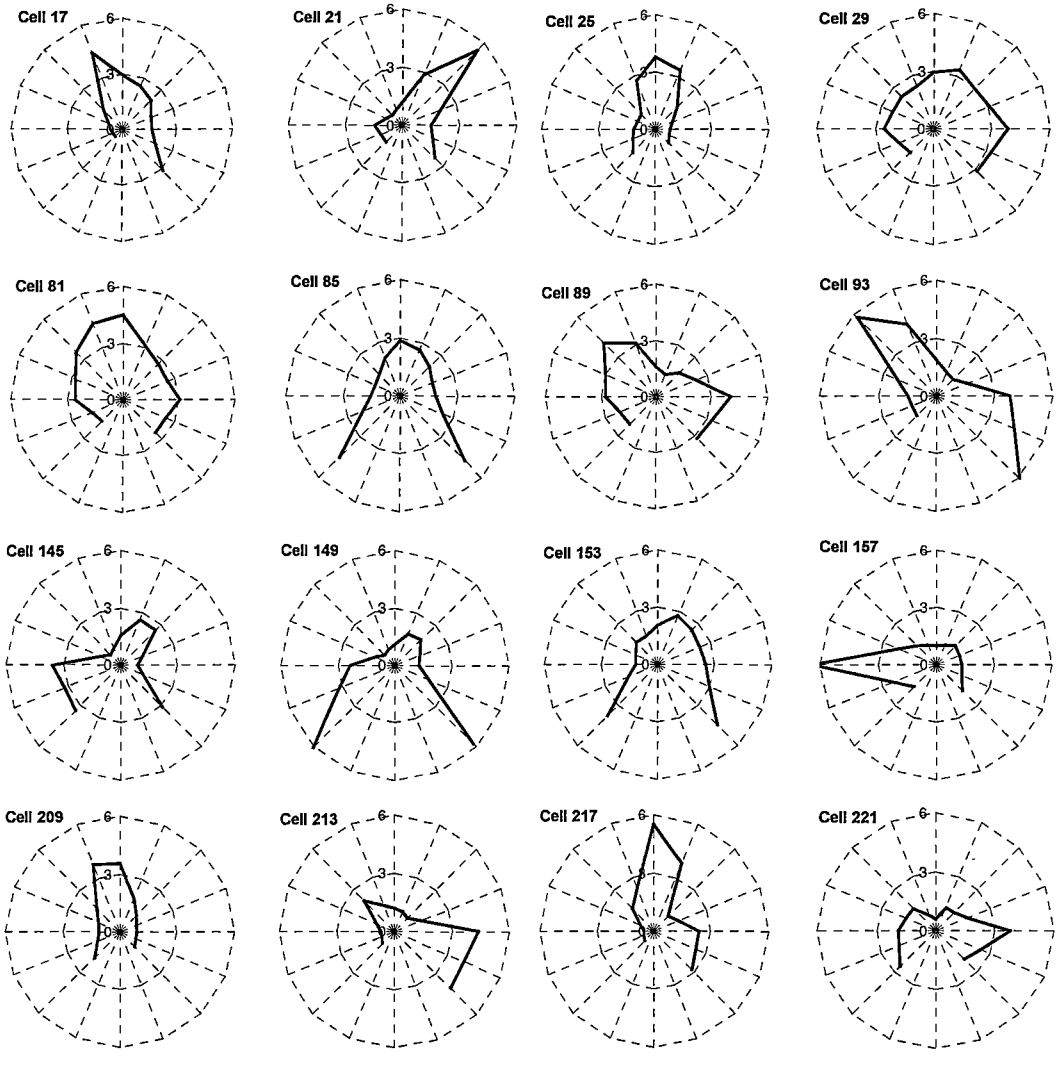

Figure 5.9: View patterns learned by 16 representative cells from an array of 225. View cell input is in the form of 9 range readings at -135, -90, -45, -22.5, 0, 22.5, 45, 90, and 135 degrees with respect to the orientation of the robot. 0 degrees (the robot heading) is directly up in the diagrams. The dotted concentric circles are at ranges of 3m and 6m.

It can be seen that the ARFNs have learned to respond to a variety of different views ranging from corridors at various orientations, to open spaces. What cannot be seen from these diagrams is the shape of the response function for each input, which is

important for understanding the range of views that each cell will respond to. Three of these view cells (25, 81 and 213) were chosen for more detailed analysis. Detailed plots of these response functions are given below in Figure 5.10, Figure 5.11 and Figure 5.12.

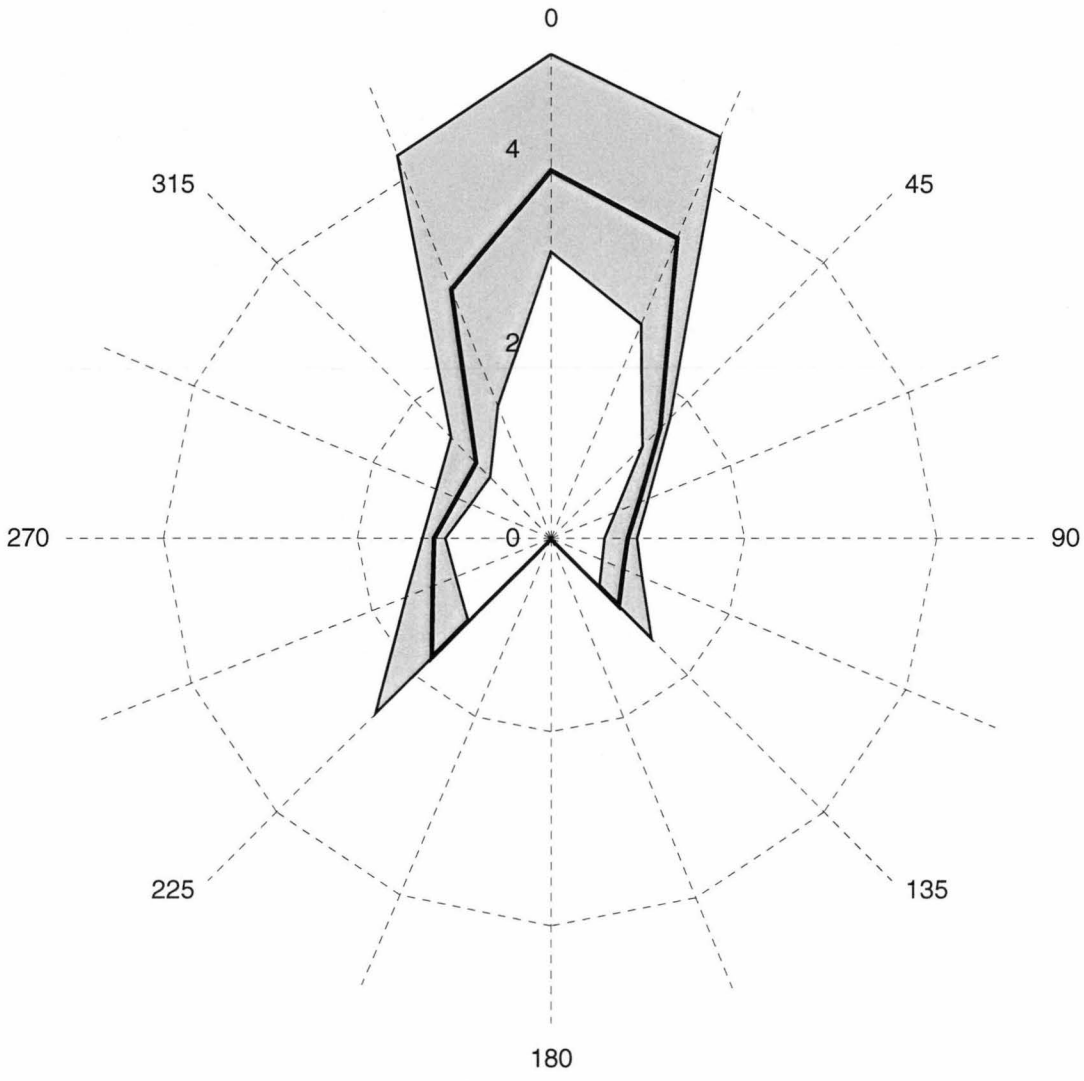

Figure 5.10: The response function learned by cell 25. View cell input is in the form of 9 sonar readings at –135, -90, -45, -22.5, 0, 22.5, 45, 90, and 135 degrees with respect to the orientation of the robot. The thick line plots the range at which each input fires maximally, while the shaded region shows the ranges at which each input response is over 0.75.

The response function for cell 25 (Figure 5.10) has learned to recognise a situation where the robot is moving down a corridor. The width of the corridor is tightly defined (small response range), while the length of the corridor is more ambiguous. Therefore, this cell should respond to corridors of a particular width but of various lengths, or to remain active as the robot moves down a particular corridor.

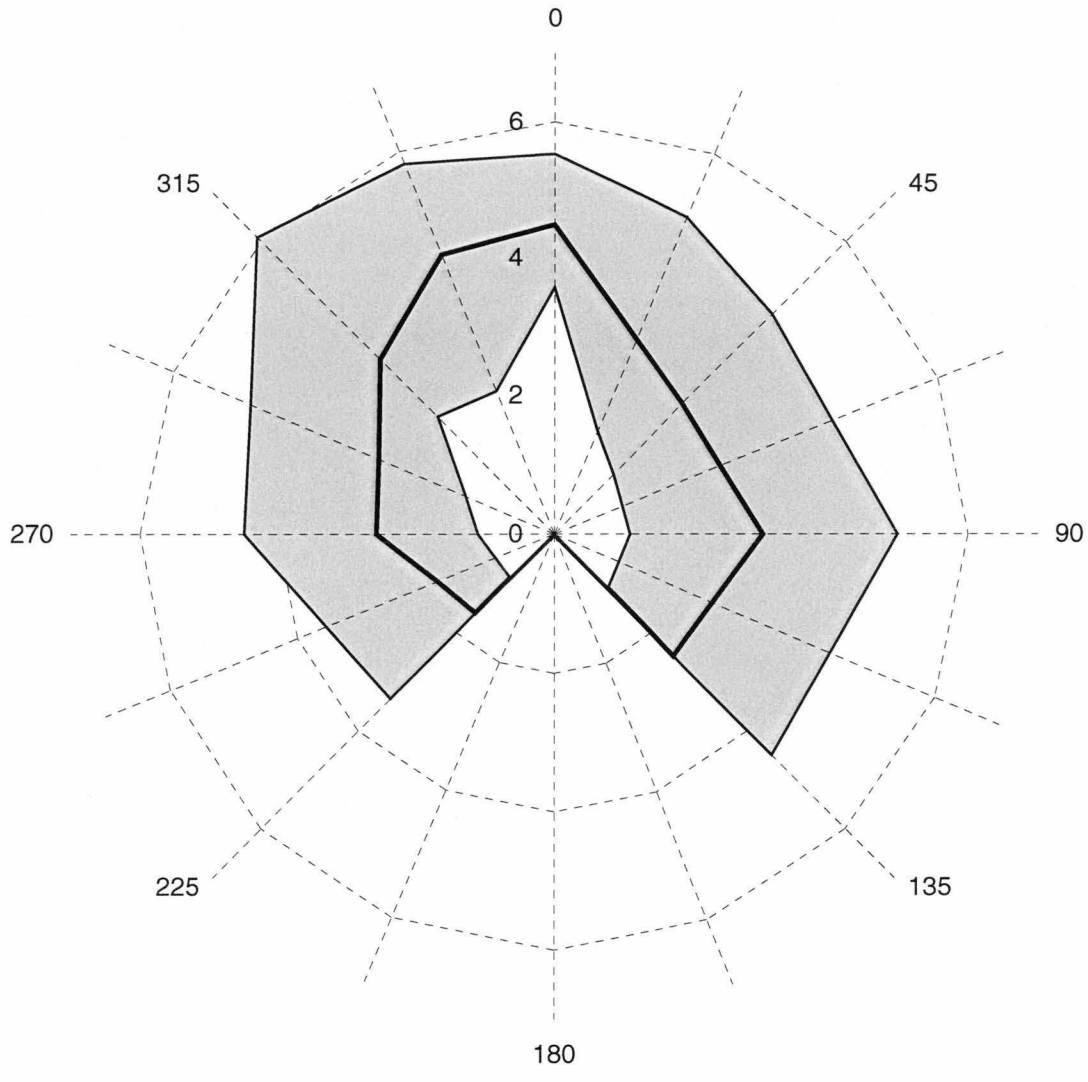

Figure 5.11: The response function learned by cell 81. View cell input is in the form of 9 range readings at –135, -90, -45, -22.5, 0, 22.5, 45, 90, and 135 degrees with respect to the orientation of the robot. The thick line plots the range at which each input fires maximally, while the shaded region shows the ranges at which each input response is over 0.75.

Cell 81 (Figure 5.11) recognises a view where the robot faces an irregularly shaped open space. The approximate diameter of the space is four metres (the size of the central area in the environment), however a reasonable response will be given for diameters between 2 and 10 meters. This range also means that the response will be high for various positions within the open space.

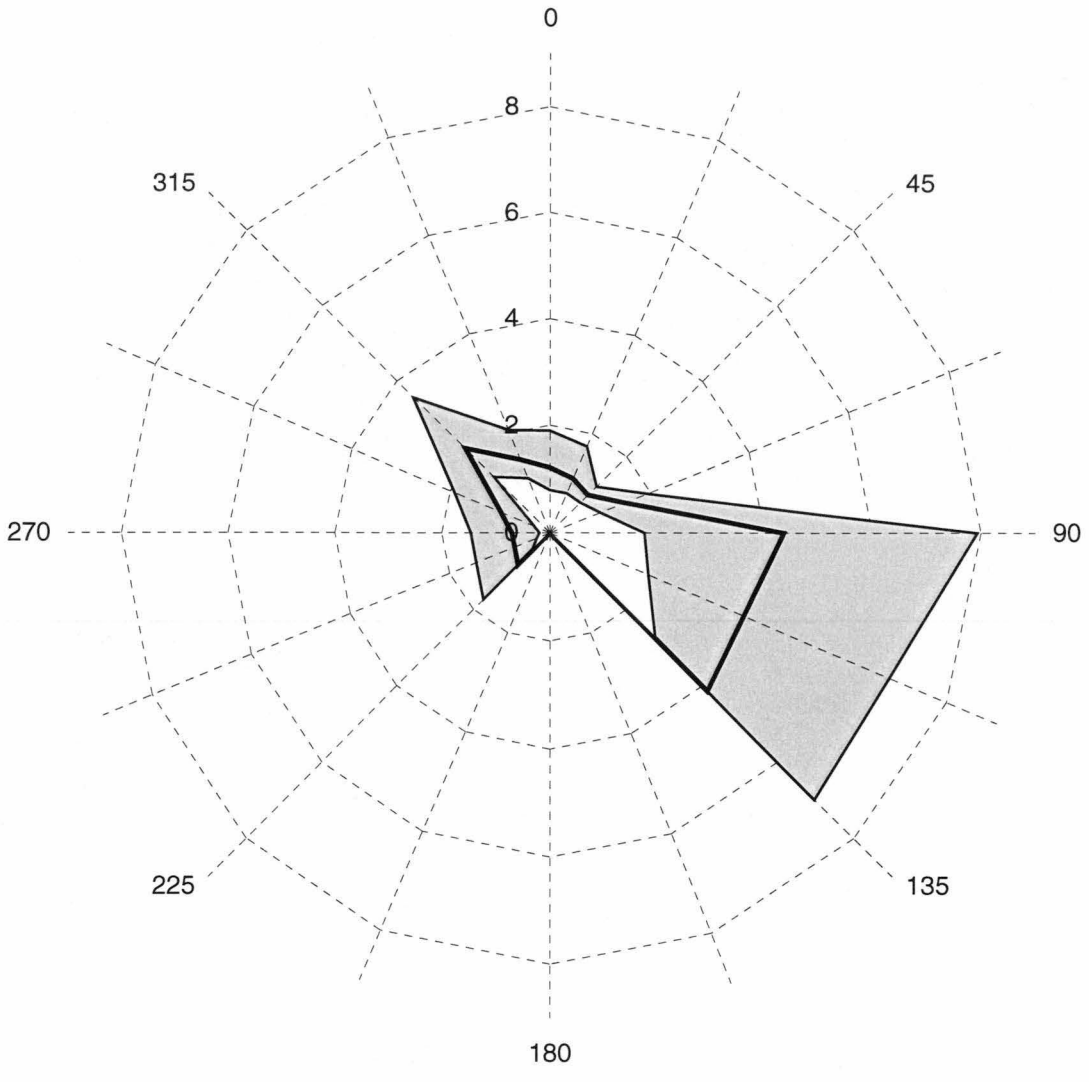

Figure 5.12: The response function learned by cell 213. View cell input is in the form of 9 range readings at –135, -90, -45, -22.5, 0, 22.5, 45, 90, and 135 degrees with respect to the orientation of the robot. The thick line plots the range at which each input fires maximally, while the shaded region shows the ranges at which each input response is over 0.75.

Figure 5.12, the response function for cell 213, depicts a view where the robot is facing towards a corner with a large open space towards the right rear. Notice that the walls near the corner are reasonably well defined whereas the size of the of the open space is not.

These view cells are able to detect a range of situations with similar salient features. Without the ability to learn variable response function widths, the view cells would be much more limited. For example in a corridor situation, a view cell with a fixed narrow response function would identify corridors of one particular length, or one particular position within the corridor. In contrast, a view cell with a broad response

function would respond to open spaces as well as corridors. Not only does this mimic the behaviour of biological view cells, which are shown to have fields that follow boundaries in the environment, but this is also likely to be a desirable property. For example, given a particular goal location, the appropriate action is likely to be the same for any position with the corridor.

While these plots give some indication of the usefulness of view cells trained in this way, further information can be gained from analysing the particular location and orientations where these cells were the most active. This information, shown in Figure 5.13, will further help to determine the suitability of these cells for place cell input.

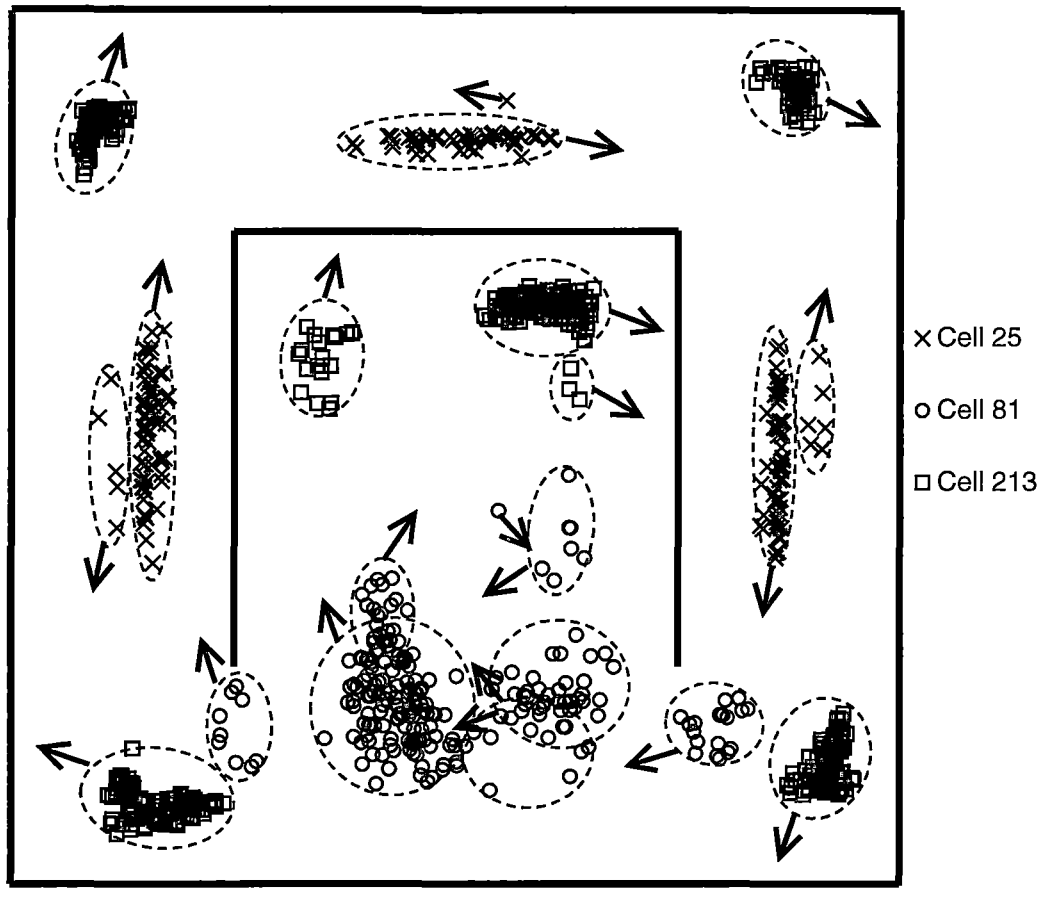

Figure 5.13: Location and orientation where the winning ARFN was either cell 25, 81 or 213. Solid lines represent walls. Dotted ovals show groups of cells sampled at similar orientations, with the average orientations indicated by arrows. The perimeter of the environment measures 8×8m.

Figure 5.13 show the locations and orientations where each of the three cells considered was the most active. For any given orientation, these view cells may

have more than one place field, however these fields are generally separated enough that they should be distinguishable through path integration, with the possible exception of cell 81.

The view fields of cell 81 are spread over a broad range of positions and orientations near the entrance to the open space. While initially this may seem problematic, it is interesting to note that biological place fields are also larger in open areas than in more restrictive parts of an environment. The overlapping place fields of other cells should help reduce ambiguity where a more restricted place code is required.

5.3. Summary

This chapter has presented a neural model, called the adaptive response function neuron, capable of learning a locally tuned response function that responds selectively to the training examples. The new model is able to adjust the centre, width and shape of the response function to match the training data in an intuitive and powerful way. The model suggests a simple architecture for the formation of locally tuned neurons in the cerebrum and other cortical areas. Networks of adaptive response function neurons may be applied and trained in the same way as radial basis function networks or self-organising maps. Adaptive response function neurons have the advantage that fewer cells are required to capture the important aspects of the input data.

A system was proposed for training adaptive response function neurons as view cells. These view cells are able to generalise between input representing the same view at slightly different orientations or positions, without losing the ability to discriminate between conceptually different views. Thus, adaptive response function view cells show a relatively high correlation to the position and orientation within an environment, and should provide an excellent source of sensory information for the establishment of place cells. The ability of these view cells to generalise between similar views should also prove useful for low-level navigational systems.

Chapter 6. From View Cells to Place Cells

The view cells produced by the system described in the previous section provide a good basis for place cell input. View cells show good place and orientation discrimination. However, they often have place fields in more than one position and orientation. If a good estimate of head direction is available, this situation is significantly improved. A path integration system that allows only those places that are within a reasonable distance of the current estimate to be recognised would be sufficient to resolve any remaining ambiguity.

Section 6.1 presents a method for combining path integrator and view cell input, and examines the place fields generated by this method. The correction of path integration errors is discussed in section 6.2.

6.1. Combining Path Integrator and View Cell Input

While an attractor model (Kali & Dayan, 2000) of path integration is a popular model for biological systems, it was decided that such a model would be computationally too expensive for the system under development. Instead, the path integrator simply stores an estimate of the robot's coordinates in the XY plane, and updates these from self-motion measurements.

Evidence from sleep experiments suggests that the relationships between cells comprising the path integration system are partially pre-configured (Kudrimoti et al., 1995). Therefore in the proposed place cell system, each cell is assigned a fixed set of path integrator coordinates. Any method may be chosen for this assignment, for the current work, the assigned coordinates correspond to a square grid of place field centres. This assignment is made with no knowledge of the environment other than the maximum size. While a random or self-organising allocation of coordinates would be biologically more plausible, given a large number of randomly allocated cells it should be possible to find a subset that approximates a square grid.

Path integrator coordinates are primarily updated from odometric estimates of the robot's change in position. The primary influence on place cell activity is based on the Gaussian distance of the centre of the cell's place field from the current path integrator coordinates.

The path integrator contribution to the activation of place cell i is given by:

$$PI_i = \exp\left(\frac{-\|\overline{p}' - \overline{p}_i\|^2}{\sigma^2}\right) \qquad \textbf{6.1}$$

where $\overline{p}'$ is the current path integration vector, $\overline{p}_i$ is the centre of place cell i's place field and σ is a parameter controlling the range of the path integrator contribution.

Odometric errors may result from undetectable occurrences such as wheel slip or collisions. These errors will cause cumulative path integration errors and must be corrected by view cell input. However, view cell input alone should not be sufficient to cause place cell firing. Therefore, view cell input is used to moderate the path integrator input, rather than drive it. The place cell system learns an association between view cell input and place cell firing. View cell input may be significantly different for different robot headings in the same place, and so a separate association is learned for each of a discrete set of orientations. During each update cycle the weight, w_{ij}^d, from view cell i to place cell j, for direction d, is adjusted using the modified Hebbian rule:

$$\Delta w_{ij}^d = \begin{cases} \eta_v PC_j(1 - PC_j)(VC_i - T_{vc}) & ,if\ d = \hat{h} \\ 0 & ,otherwise \end{cases} \qquad \textbf{6.2}$$

where $\hat{h}$ is the discretised value of the current heading, h; VC_i is the output of view cell i; PC_j is the output of place cell j; T_{vc} is a threshold parameter determining the effect of view cell activation on the direction of weight changes; and η_v is the training rate. If the current heading is $\hat{h}$, the view cell input, VI, to place cell j is given by:

$$VI_j = \frac{2}{1 + \exp\left(-\sum_i w_{ij}^{\hat{h}}(VC_i - T_{vc})\right)} - 1 \qquad \textbf{6.3}$$

If view cell output is greater than T_{vc}, this view cell will contribute positively to place cell firing, otherwise it will inhibit place cell firing. The final place cell output is given by:

$$PC_j = \frac{1}{1+\exp\left[t-\left(aPI_j+bVI_j\right)\right]} \qquad \textbf{6.4}$$

where the parameters t, a and b are chosen so that view cell input alone does not produce significant place cell activation, as shown in Figure 6.1. For the values of t, a, and b, chosen in Figure 6.1, path integration input alone will produce moderate out put enabling the system to learn weights for view cell input. Once view cell connection weights are established, input from these cells pushes place cell activation higher, but is not large enough to produce high activation if no path integration input is present. A higher value of b would result in problems with perceptual aliasing, since similar views may exist in different parts of the environment. If path integration input is present and view cell input, previously correlated with that location, is not present, the view cell contribution will significantly reduce place cell activity, indicating a path integration error.

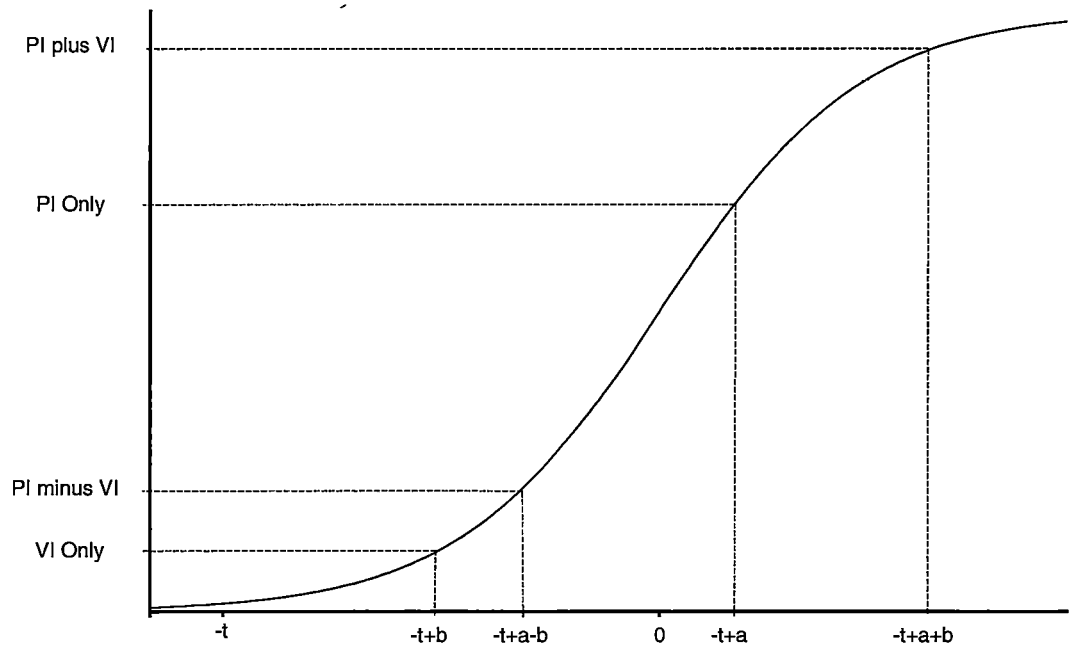

Figure 6.1: Choosing appropriate parameters, a, b, and t, for the contributions of view cell and path integration input to place cell firing. Dashed lines show the activation levels for (from lowest to highest) strong view cell input only, strong path integration input with strong negative view cell input, strong path integration input only, and strong path integration and view cell input.

6.1.1. Place Fields

The place fields generated in this way should show a high degree of positional selectivity. The shapes of place fields will also be influenced by the current view and hence the orientation of the robot. Figure 6.2 shows the place fields of nine place cells sampled during a collision avoidance task.

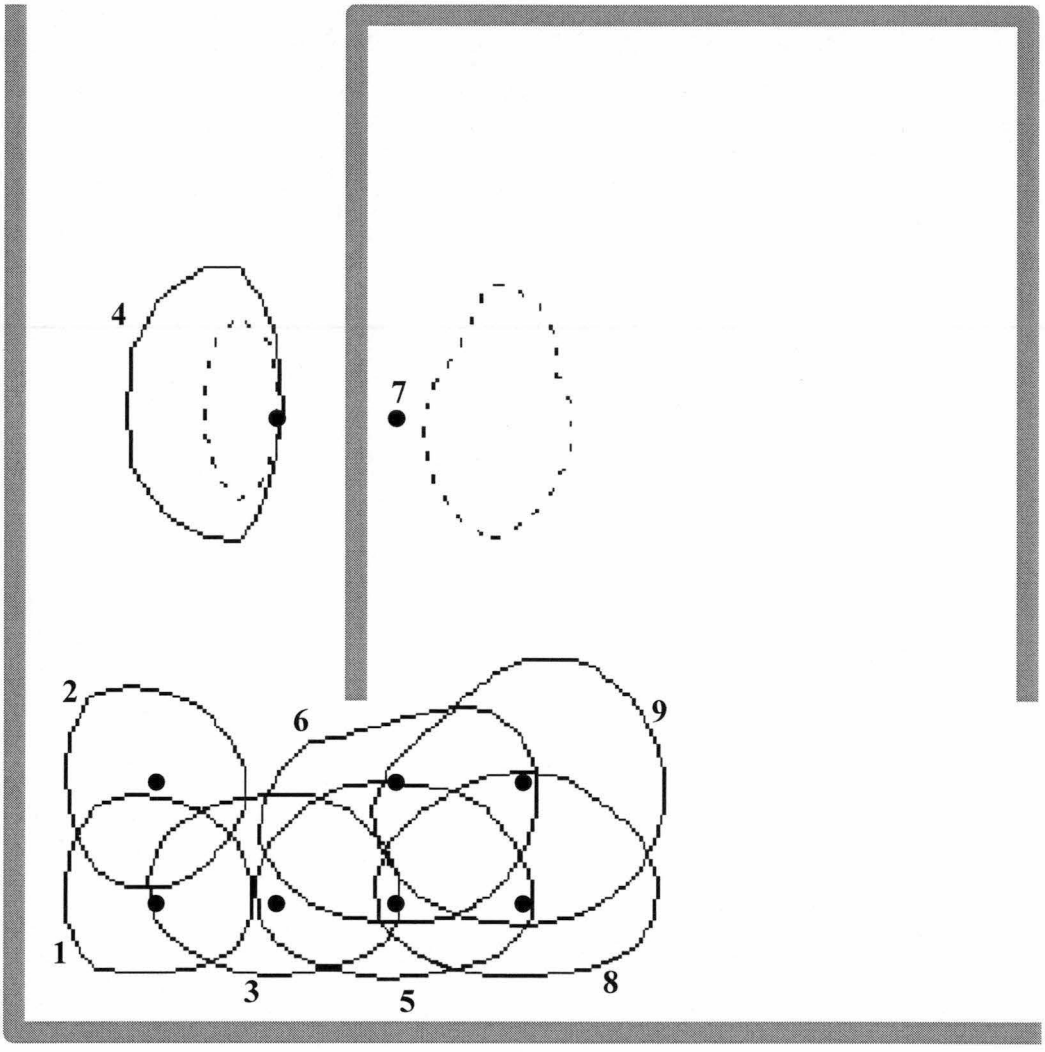

Figure 6.2: Place fields of 9 place cells sampled in the southwest corner of the environment. Data was averaged over all robot orientations. The place cell path integrator coordinates are indicated by small solid circles. Solid contours indicate an activation level of 0.5, and are shown for 9 cells (Cells 1-6,8,9). The dotted contours indicate the 0.25 activation level of a single bimodal place cell (Cell 7).

The generated place fields show a high degree of overlap, which would provide good generalisation for any navigational system based on these cells. The shapes of place fields also conform to the environment. For example, the field of cell 4 is elongated in the direction of the corridor, and the field of cell 9 bends around the corner of the

wall. The field of cell 7 is clearly bimodal. However, the activation levels of this cell were significantly lower than other cells. A more detailed analysis of four of these fields follows in Figure 6.3, Figure 6.4, Figure 6.5, and Figure 6.6.

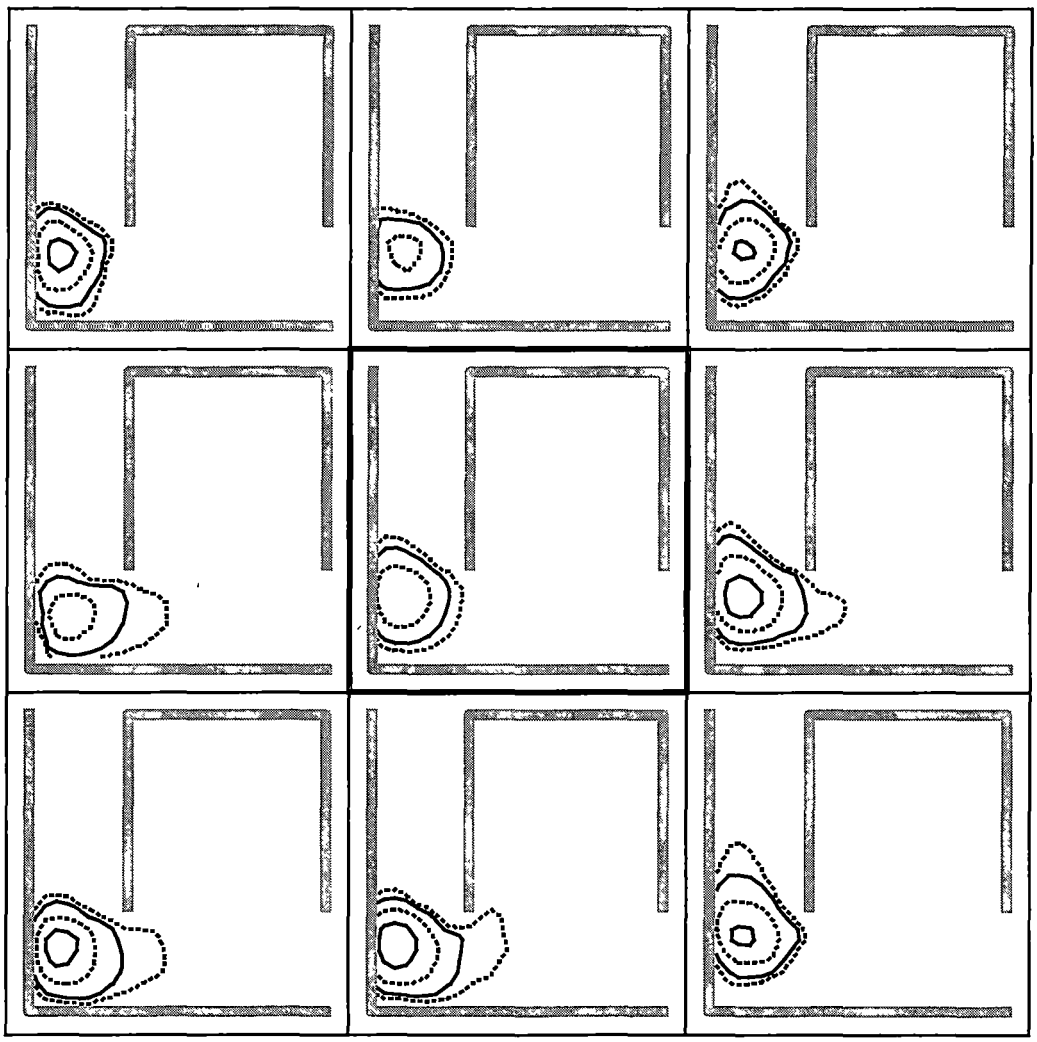

Figure 6.3: Place field detail for cell 1. Dotted contours indicate activation levels of 0.15 and 0.5. Solid contours indicate activation levels of 0.3 and 0.7. The central figure shows the average activation over all robot headings, while the surrounding figures show the activation average of headings within 22.5 degrees of each compass point.

The path integrator coordinates for cell 1 correspond to a location near the southwest corner of the environment. For all orientations, the place field centre is located close to these coordinates. However, the shape of the place field varies significantly with orientation. It is not clear in this instance whether this variation would be enough to adversely affect navigation, or conversely whether this distortion may in fact be beneficial.

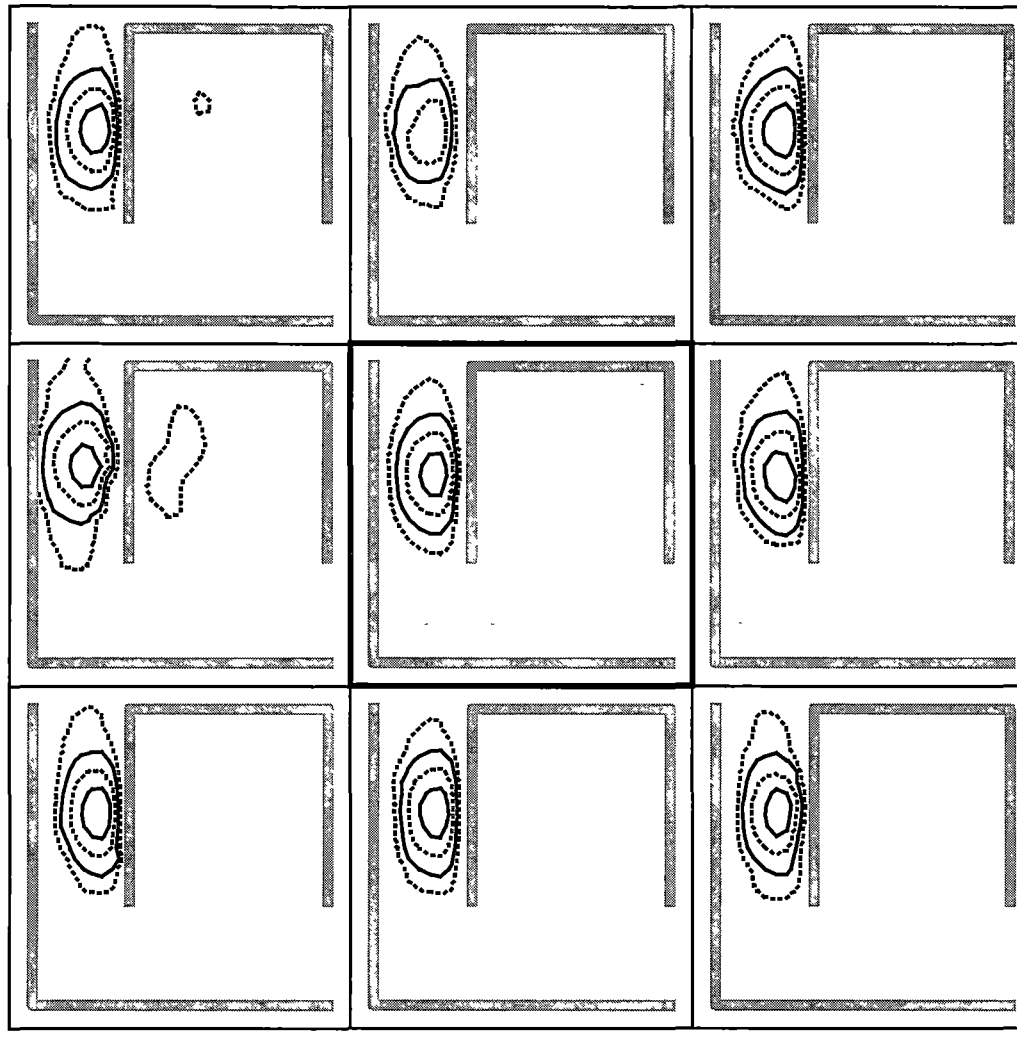

Figure 6.4: Place field detail for cell 4. Dotted contours indicate activation levels of 0.15 and 0.5. Solid contours indicate activation levels of 0.3 and 0.7. The central figure shows the average activation over all robot headings, while the surrounding figures show the activation average of headings within 22.5 degrees of each compass point.

The field of cell 4 is elongated along the corridor and, since the appropriate action is not likely to change in this region, this should be a beneficial property. For orientations to the northwest and west, this cell shows some bimodal behaviour. However, the activity level in the secondary field is very low, and not likely to affect navigation.

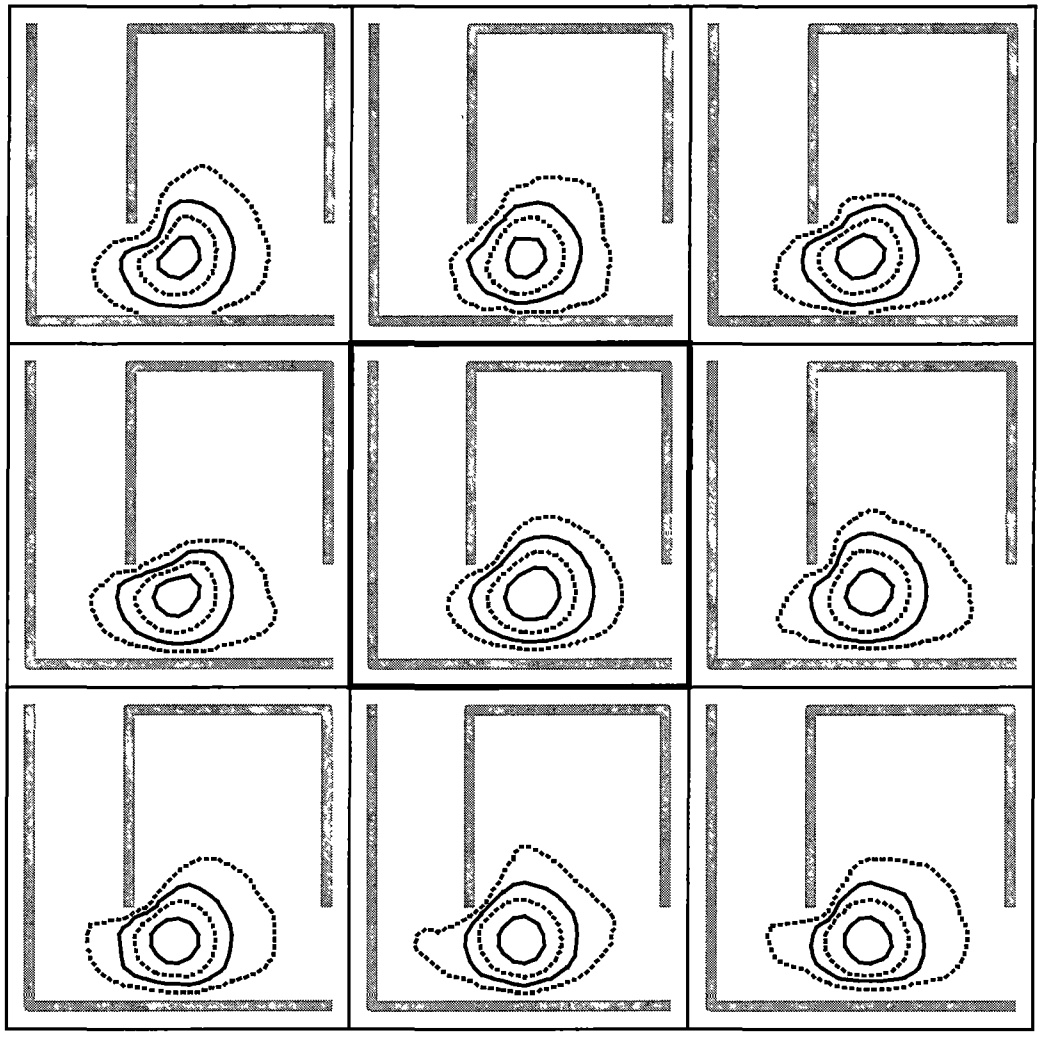

Figure 6.5: Place field detail for cell 9. Dotted contours indicate activation levels of 0.15 and 0.5. Solid contours indicate activation levels of 0.3 and 0.7. The central figure shows the average activation over all robot headings, while the surrounding figures show the activation average of headings within 22.5 degrees of each compass point.

Cell 9 has a place field that also shows some variation with the orientation of the robot. However, the area of peak activity is quite stable with respect to orientation, and should not pose a problem for the navigational system. The size of this cell's place field is also larger than for those cells in a more restricted part of the environment, and this is in agreement with experimental results for behavioural studies.

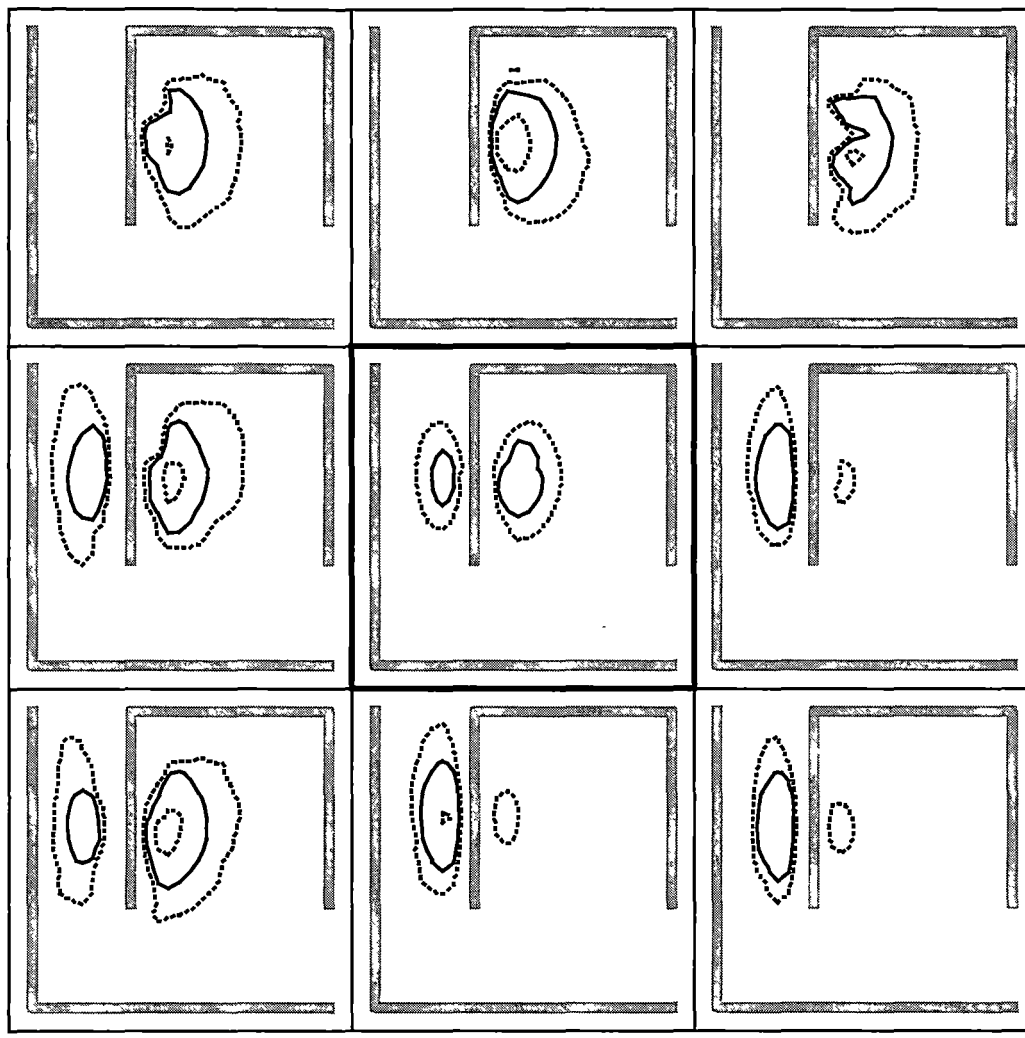

Figure 6.6: Place field detail for cell 7. Dotted contours indicate activation levels of 0.15 and 0.5. Solid contours indicate an activation level of 0.3. The central figure shows the average activation over all robot headings, while the surrounding figures show the activation average of headings within 22.5 degrees of each compass point.

Cell 7 has a place field with a distinctly bimodal nature. In addition, the area of greatest activity is dependent upon the orientation of the robot. Furthermore, the two centres of activity are located on opposite sides of the wall. This cell would not be suitable as input to a navigational system. However, the maximum output of this cell is considerably lower than for other cells and in fact the output of this cell was always dominated by neighbouring cells such as cell 4. Hence, even this distinctly bimodal cell will not have an adverse effect on navigation.

6.2. Correcting Odometric Errors

The place fields generated by this algorithm show many of the properties of biological place fields, and should provide valuable input to the navigational system. In addition to navigational input, the place cells should also be able to correct for odometric errors in the path integration system on which they rely for input. To correct the position estimate, an estimate of the robot's current location is calculated as the average of place field centres, weighted by the view moderated place cell output. The difference between this population vector (Georgopoulos et al., 1988) and the current position estimate is calculated, and the position estimate is updated using:

$$\Delta\overline{p} = \eta_p \left(\frac{\sum_i PC_i \overline{s}_i}{\sum_i PC_i} - \overline{p} \right) \qquad \textbf{6.5}$$

where η_p is the training rate. Note that if view cell to place cell weights are low, as when the robot first enters the environment, $\Delta\overline{p}$ will be very small. That is, the robot will initially trust it's path integrator coordinates.

This process is best illustrated by an example. Figure 6.7 shows a typical situation where the robot approaches a wall after having accumulated an error in the path integrator coordinates.

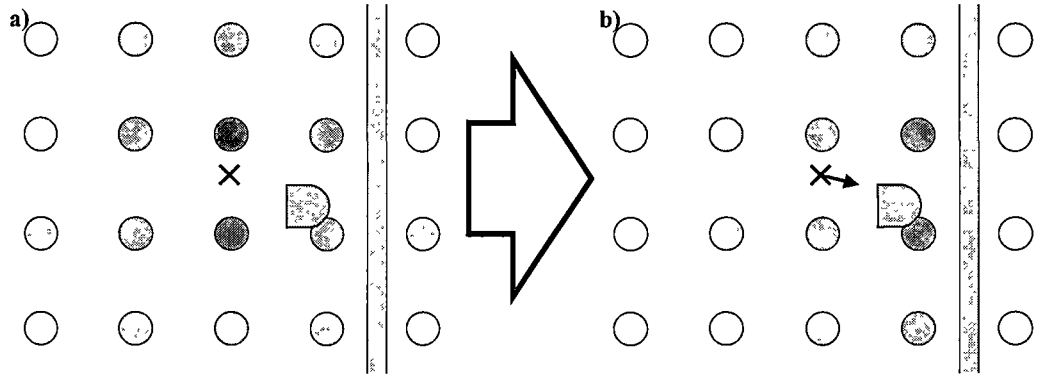

Figure 6.7: The influence of path integration and view cell input on place cell activity. As the robot approaches a wall the path integration coordinates, indicated by an 'X', are currently incorrect and produce the place cell activity, indicated by shaded circles, shown in a). However, the current head direction and view will be more associated with place cells that are close to, and to the left of, the wall. These place cells will have their activity increased, whereas other place cells will have their activity decreased. This view cell moderated activity, shown in b), results in a new position estimate and the path integrator coordinates are shifted towards this new value.

The ability of the place cell system to correct for path integration errors was tested by adding noise to the robot's path-integration estimate, as well as a small systematic error at each time step. This error would cause the position estimate to drift if not corrected. If place cells were distributed over an area the same size as the environment then the position estimate would be easily corrected by the system as the edges of the environment were approached. To remove the possibility that edge effects could unfairly allow the system to correct errors, place cells were distributed over an area significantly larger than the accessible environment. Results are shown in Figure 6.8.

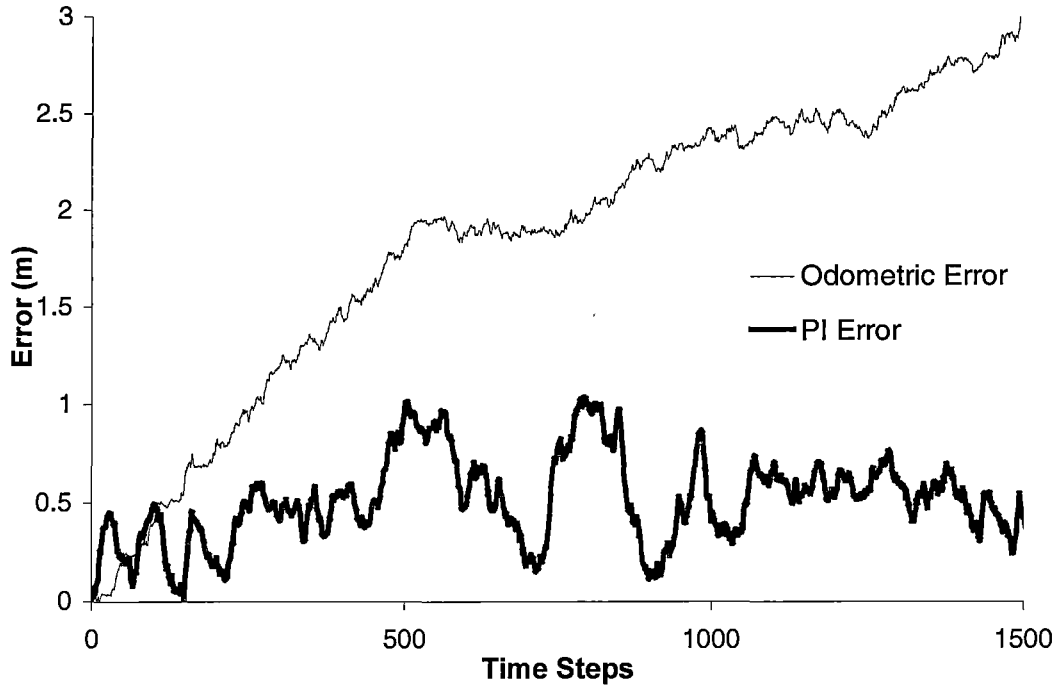

Figure 6.8: Error in position estimate over time. The thin line shows the cumulative odometric error. The thick line shows the error in the path integration estimate.

Due to the coarse nature of the place cell distribution in these experiments, the position estimate is quite noisy, but importantly the error in this estimate does not increase when self-motion estimates are systematically incorrect. This amount of variance is generally not a problem for the types of navigational problems investigated, and minor errors may be partially countered by the collision avoidance sub-system (see section 7.2). In a biological system, the vastly increased numbers of place cells would result in a much more precise estimate.

6.3. Summary

The place cell system developed is able to maintain a reasonably accurate estimate of the robot's position even in the presence of random and systematic odometric errors. The system is relatively easy to implement and the implementation is computationally inexpensive. The place fields generated show many of the properties of biological place fields, and in most cases fields are quickly learned that are unambiguous with respect to the environment. While some of the generated place fields are bimodal, the activity levels of these cells are considerably lower than other cells. Therefore, these cells are unlikely to cause problems for the navigational system.

Similar place cell systems have previously been developed. The main advantage of this system, however, is that place field centres are fixed prior to training. This allows downstream navigational systems to make a priori assumptions about the relative positions of each place cell's place field. In particular, it should prove useful to assume an open environment and initialise the navigational system accordingly. This mechanism may help explain the dead-reckoning abilities of some animals in open environments. However, even in complex environments, such an initialisation will provide a reasonable first guess for the best direction in which to travel. This issue is discussed further in section 9.2.

One disadvantage of the system is that it is unable to perform absolute localisation. That is, localisation without the benefit of odometric information, such as when the robot is first placed in a known environment at an unknown location, or when the robot is lifted and moved to a new location. The proposed system is unable to maintain multiple estimates of its current position. Therefore, in the absence of reliable odometry, it is forced to chose one location that best matches the current view. In environments where perceptual aliasing may occur, a poor first estimate may lead to an unrecoverable situation. This problem could be easily solved by maintaining multiple position estimates instead of just one, or by implementing an attractor model. Scenarios requiring absolute localisation were not investigated in the current work.

Chapter 7. Low-Level Navigation

This chapter develops a system for low-level navigation. The goal of this system is to allow the robot to perform rudimentary exploration of an unknown environment, and avoid collisions. A common technique for learning autonomous behaviour in mobile robots is reinforcement learning (e.g. Asada, Noda, Tawaratsumida, & Hosoda, 1996; Gaskett, Fletcher, & Zelinsky, 2000; Mahadevan & Connel, 1992), and this fits well with the biological motivations of the current work.

Reinforcement learning is the process of learning the appropriate action for a given situation or state, based solely on the experienced reward. This is in contrast to supervised learning, where the appropriate action is given to the learning agent by some external knowledgable entity, and the agent must learn to reproduce that action. While supervised learning is a powerful learning/teaching technique, in many situations it is not possible to obtain examples of suitable behaviour, or access to a knowledgable supervisor.

In the context of navigation, the current state is typically the perceived location of the agent, the action is the direction of movement, and a reward is experienced when a goal location is reached. In this type of problem, the agent will wander around the environment for some time, finally reaching the goal by some circuitous route. This presents two important problems to the learning agent.

The first problem is that of temporally distal rewards. It is difficult to learn about the action just performed if no feedback is given. When the goal location is reached, the action just performed may be preferred when in the same previous location in future, but a mechanism should also exist for learning about actions and locations prior to the most recent action.

The second problem is that of exploration versus exploitation. If all actions leading up to the goal location are to be chosen in preference to others, then every route to the goal will follow the same path as the first successful attempt. However, this path may be far from optimal, and some attempt should be made to find shorter paths. Obviously, at some point the agent must choose the optimal action in preference to exploratory choices. The difficulty is in finding the right balance, and this often depends on the type of problem being solved.

Section 7.1 introduces the temporal difference learning algorithm. This reinforcement learning algorithm is one popular solution to the problem of

temporally distal rewards. Also discussed in this section are some strategies for addressing the exploration versus exploitation problem. Section 7.2 demonstrates the use of temporal difference learning for low-level navigation.

7.1. Temporal Difference Learning

Temporal difference (TD) learning (Sutton, 1988) is a popular reinforcement learning method that updates the value of the action just performed based on the estimated value of the current state. The value of a state, with respect to a particular action selection policy, is the expected discounted future reward if that policy is followed from the current state onwards. That is, if the agent receives reward r_t at time t, then for a policy π and discounting factor γ ($0<\gamma<1$), the value $V^{\pi}(s_t)$ of state s_t is:

$$V^{\pi}(s_t) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+1+k}\right\} \qquad \textbf{7.1}$$

This can be rewritten in terms of the value of the subsequent state:

$$\begin{aligned} V^{\pi}(s_t) &= E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+1+k}\right\} \\ &= E_{\pi}\left\{r_{t+1} + \sum_{k=1}^{\infty}\gamma^k r_{t+1+k}\right\} \\ &= E_{\pi}\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{(t+1)+1+k}\right\} \\ &= E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1})\right\} \end{aligned} \qquad \textbf{7.2}$$

Equation 7.2 can be used to derive a rule for learning value estimates on line. The error, δ, in the estimated value, V, of state s_t is defined as:

$$\delta = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \qquad \textbf{7.3}$$

If the training rate is η, then the value estimate is updated using equation 7.4:

$$V(s_t) \leftarrow V(s_t) + \eta\delta \qquad \textbf{7.4}$$

The full algorithm is given below in Figure 7.1.

Initialise $V(s) \forall s, \pi$
Initialise s
Repeat:
$a \leftarrow$ action given by π for s
Take action a; observe reward r, and next state, s'
$\delta \leftarrow r + \gamma V(s') - V(s)$
$V(s) \leftarrow V(s) + \eta\delta$
$s \leftarrow s'$

Figure 7.1: The temporal difference learning algorithm.

The TD-learning algorithm partially solves the problem of learning state values from temporally distal rewards, but the values are only valid for the current policy. The next section presents a method for learning the optimal policy and value function simultaneously.

7.1.1. Actor-Critic

One of the earliest implementations of TD-learning for action selection was the actor-critic architecture (Barto et al., 1983). When using TD-learning, if the error in a value estimation, δ from equation 7.3, is positive, the action just performed was more favourable than expected. The preference for choosing that action when encountering the same situation in future should be increased, and the converse is true for negative actions. If the preference for choosing action a_t from state s_t is $p(s_t, a_t)$, then these preferences may be updated with training rate η' using:

$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \eta'\delta \qquad \textbf{7.5}$$

A suitable policy, π, may then be constructed based on these preferences, and this policy may also address the exploration versus exploitation problem. For example, the ε-greedy selection policy chooses the most preferable action on most occasions, but occasionally, with probability ε, chooses an alternate exploratory action. Alternatively action probabilities, $\mathcal{P}$, may be derived from a Boltzman distribution with 'temperature' τ:

$$\mathcal{P}(s,a) = \frac{e^{p(s,a)/\tau}}{\sum_b e^{p(s,b)/\tau}} \qquad \textbf{7.6}$$

Through the TD-learning algorithm and appropriate action selection strategies, the actor-critic architecture is able to solve both of the problems confronting a reinforcement learning agent. In addition, this architecture fits well with some recent models of biological reinforcement learning in the basal ganglia (see section 2.4 for details). However, an alternate approach, where action values for each state are explicitly represented, is often preferred.

7.1.2. SARSA

The SARSA (state-action, response, state-action) algorithm (Rummery & Niranjan, 1994) uses the TD technique to learn the action value function, Q, directly. The error in an action value prediction $Q(s_t, a_t)$ for an action a_t performed at time t from state s_t in this modified algorithm is:

$$\delta \leftarrow r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \qquad 7.7$$

By applying the update rule in the normal way, action values are learned for the current policy. That policy is usually derived from the action values themselves, using techniques such as those used for the action preferences in the actor-critic architecture. Hence, the policy and value functions are learned simultaneously. This is known as on-policy learning.

This simple algorithm, shown in Figure 7.2, often performs better than the slightly more complex actor-critic architecture, but as with actor-critic, the policy learned is not necessarily the optimal policy.

Initialise $Q(s, a) \forall s, a$
Initialise s
Choose a from s using policy derived from Q
Repeat:
    Take action a; observe r, s'
    Choose a' from s' using policy derived from Q
    $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
    $Q(s, a) \leftarrow Q(s, a) + \eta\delta$
    $s \leftarrow s'; a \leftarrow a'$

Figure 7.2: The SARSA learning algorithm.

7.1.3. Q-Learning

Q-learning (Watkins, 1989; Watkins & Dayan, 1992) is an off-policy method, meaning the value function learned is for the optimal policy, regardless of the policy currently being followed. This is achieved by using the maximum action value from the current state to train the value for the previous action, regardless of which action is actually chosen from the current state. That is:

$$\delta \leftarrow r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \qquad \textbf{7.8}$$

It has been proven that, provided each state-action pair is visited equally often, Q-learning converges to the optimal action value function (Watkins & Dayan, 1992), and hence the optimal policy may be derived. The complete algorithm is given in Figure 7.3.

Initialise $Q(s, a) \forall s, a$
Initialise s
Repeat:
    Choose a from s using policy derived from Q
    Take action a; observe r, s'
    $\delta \leftarrow r + \gamma \max_{a'} Q(s', a') - Q(s, a)$
    $Q(s, a) \leftarrow Q(s, a) + \eta\delta$
    $s \leftarrow s'$

Figure 7.3: The Q-Learning algorithm.

7.1.4. Eligibility Traces

All of the TD-learning methods discussed so far allow value functions to be updated immediately after each experience, based on the estimated value of the current state. While this partially solves the problem of temporally distal rewards, in practice it may take some time before useful information is available for these updates to be meaningful. For example, consider an agent moving through states A, B, C, D, E and F to a goal state G. If the value function is initialised with zero values, then the error perceived after each state transition, A→B, B→C, C→D, D→E and E→F will be zero and no updates will be made to the value function.

After the transition F→G, the value for state F will be updated, but even now, the values for the other states will not. If after reaching G the agent returns to A, and

again follows the same path, the value for state E will be updated, since F now has a meaningful value estimate. On the next traversal, D will be updated, and so on. What is needed is a mechanism to update all previously visited states after the first traversal. This mechanism is provided by eligibility traces, originally proposed by Klopf (1972).

An eligibility trace *e(s)* is maintained for each state. If replacing traces are used, this trace is set to 1 when the state is visited and decays by $\lambda\gamma$ ($0<\lambda<1$) at each time step. That is:

$$e_t(s) = \begin{cases} 1, & \text{if } s_t = s \\ \lambda\gamma e_{t-1}(s), & \text{otherwise} \end{cases} \qquad \textbf{7.9}$$

Alternatively, accumulating traces may be used:

$$e_t(s) = \begin{cases} e_t(s) + 1, & \text{if } s_t = s \\ \lambda\gamma e_{t-1}(s), & \text{otherwise} \end{cases} \qquad \textbf{7.10}$$

For the type of tasks being examined, replacing traces offer a significant improvement over accumulating traces (see Sutton & Barto, 1998, p186 for details). Therefore, replacing traces were used for all experiments.

TD(λ) (TD-learning using eligibility traces) updates all state values at each time step based on their eligibility trace. The error is calculated using equation 7.3 and each state value is updated using:

$$V(s) \leftarrow V(s) + \eta e(s)\delta \quad ,\forall s \qquad \textbf{7.11}$$

The complete algorithm is given in Figure 7.4.

Initialise $V(s)\ \forall s; \pi$; and $e(s) = 0\ \forall s$
Initialise s
Repeat:
    $a \leftarrow$ action given by π for s
    Take action a; observe reward r, and next state, s'
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $e(s) \leftarrow 1$
    for all s:
        $V(s) \leftarrow V(s) + \eta \delta e(s)$
        $e(s) \leftarrow \gamma \lambda e(s)$
    $s \leftarrow s'$

Figure 7.4: The Temporal Difference learning algorithm with the addition of eligibility traces.

Similarly for SARSA(λ), an eligibility trace, *e(s,a)*, is maintained for each action pair and updated using:

$$e_t(s,a) = \begin{cases} 1, & \text{if } s_t = s,\ a_t = a \\ \lambda\gamma\, e_{t-1}(s,a), & \text{otherwise} \end{cases} \qquad \textbf{7.12}$$

The action value function is updated using:

$$Q(s,a) \leftarrow Q(s,a) + \eta e(s,a)\delta \quad ,\forall s,a \qquad \textbf{7.13}$$

Implementation of eligibility traces for Q-learning is not so straightforward. Two methods have been proposed: Watkins' $Q(\lambda)$ (Watkins, 1989; Watkins & Dayan, 1992) and Peng's $Q(\lambda)$ (Peng & Williams, 1996). Peng's $Q(\lambda)$ is not an entirely off-policy method (Sutton & Barto, 1998, p184), therefore only Watkins' $Q(\lambda)$ will be discussed here.

Watkins' $Q(\lambda)$, see Figure 7.6, cuts off eligibility traces when a non-optimal action is chosen, for example when an exploratory move is made. This is because it can not be guaranteed that the error for such an action is applicable to previous action choices, as shown in Figure 7.5.

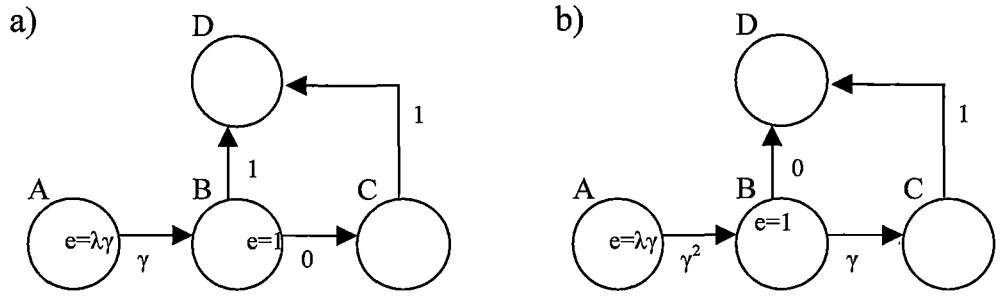

Figure 7.5: Applying eligibility traces to Q-learning. a) The values for actions from A-B, B-D and C-D (γ, 1 and 1 respectively) have been previously learned (for D as the goal). The value for action B-C is currently 0. The actions A-B and B-C have just been performed giving eligibility traces of $\lambda\gamma$ and 1 respectively. b) The values for actions from A-B, B-C and C-D (γ^2, γ and 1) have been previously learned (for D as the goal). The value for action B-D is currently 0. The actions A-B and B-D have just been performed giving eligibility traces of $\lambda\gamma$ and 1 respectively.

The error for action B-C (a non-optimal action) in Figure 7.5a is γ (from equation 7.8). The value of action B-C will correctly be trained towards this value. However, if an update were made for action A-B, based on the eligibility trace, the value of this action would be trained too high. The value for action A-B would be increased from γ towards $\gamma+\lambda\gamma\gamma$ (applying equation 7.13), when in fact the current value is correct. Resetting eligibility traces when a non-optimal action is chosen avoids this problem.

The replacing eligibility trace update rule for Watkins' $Q(\lambda)$ is therefore:

$$e_t(s,a) = \begin{cases} 1, & \text{if } s_t = s,\ a_t = a \\ \lambda\gamma\, e_{t-1}(s,a), & \text{if } s_t \neq s \vee a_t \neq a,\ a_t = \arg\max_a Q(s_t,a) \\ 0, & \text{otherwise} \end{cases} \qquad \textbf{7.14}$$

The action value update, when using Watkins' $Q(\lambda)$ is the same as in equation 7.13.

Initialise $Q(s,a); e(s,a)\ \forall s, a,$
Initialise s, a
Repeat:
Take action a; observe r, s'
Choose a' from s' using policy derived from Q
$a^* \leftarrow \arg\max_b Q(s', b)$
$\delta \leftarrow r + \gamma Q(s', a^*) - Q(s,a)$
$e(s,a) \leftarrow 1$
for all s, a:
$Q(s,a) \leftarrow Q(s,a) + \eta\delta e(s,a)$
if $a' = a^*$
$e(s,a) \leftarrow \gamma\lambda e(s,a)$
else
$e(s,a) \leftarrow 0$
$s \leftarrow s';\ a \leftarrow a'$

Figure 7.6: The Watkin's $Q(\lambda)$ algorithm.

Figure 7.5b shows another situation where action values could be updated incorrectly if eligibility traces were not reset to zero when non-optimal actions are chosen. The error for action B-D in Figure 7.5b is 1 and, using Q-learning, the value for this action will be trained towards this value. However, if the value for action A-B were updated using the eligibility trace, the calculated error would be too high. Unfortunately, resetting the eligibility trace to zero causes no training to take place at all, when in fact the value for action A-B should be increased from γ^2 to γ. The Concurrent Q-Learning algorithm developed in section 8.2 addresses this issue.

7.1.5. Function Approximation

So far, all TD-learning algorithms considered have been based on a table look-up implementation. While the table-based approach is easy to implement and comprehend, in many cases such an implementation will be unfeasible or have poor performance. If the state space is very large then implementation may be limited by the available memory. Also, many states may be quite similar, and an ability to generalise between these states will probably be desirable. A table-based approach will be unable to deal with these difficulties.

The alternative is to approximate the value function from measurements or features of the state. If $V(s)$ is a smooth differentiable value function with parameters $\bar{\theta}$, then it is possible to apply gradient descent techniques to the problem. For TD the error is calculated in the usual way using equation 7.3, and the parameters of the approximation function are updated using:

$$\bar{\theta}_{t+1} = \bar{\theta}_t + \eta \delta \nabla_{\bar{\theta}_t} V_t(s_t) \qquad \textbf{7.15}$$

If eligibility traces are used, the trace should be updated using:

$$\bar{e}_t = \gamma \lambda \bar{e}_{t-1} + \nabla_{\bar{\theta}_t} V_t(s_t) \qquad \textbf{7.16}$$

with value updates as in Figure 7.7.

Initialise $\bar{\theta}$; π; and $\bar{e} = 0$
Initialise s
Repeat:
$a \leftarrow$ action given by π for s
Take action a; observe reward r, and next state, s'
$\delta \leftarrow r + \gamma V(s') - V(s)$
$\bar{e} \leftarrow \lambda \gamma \bar{e} + \nabla_{\bar{\theta}} V(s)$
$\bar{\theta} \leftarrow \bar{\theta} + \eta \delta \bar{e}$
$s \leftarrow s'$

Figure 7.7: The temporal difference learning algorithm using the gradient descent method and eligibility traces.

For SARSA and Q-learning, action values may be learned in the same way.

In the case of linear function approximation, these rules are quite straightforward to implement. If $\bar{\phi}_s$ is a set of n measurements of state s, the value function will be of the form:

$$V_t(s) = \sum_{i=1}^{n} \theta_t(i) \phi_s(i) \qquad \textbf{7.17}$$

The gradient of the value function will then be simply:

$$\nabla_{\bar{\theta}_t} V_t(s_t) = \bar{\phi}_s \qquad \textbf{7.18}$$

Linear function approximation may be combined with various forms of feature extraction, such as self-organising maps (Kohonen, 1995), radial basis function networks (Broomhead & Lowe, 1988), and tile coding (Albus, 1971).

7.2. Low-Level Design and Testing

This section develops the low-level navigational system and addresses key design issues to be decided in the development of a reinforcement learning system. The system is then tested in the simulated environment (Appendix A), with a particular emphasis on appropriate input for this system.

7.2.1. Reward Structure

For collision avoidance, the most obvious reward structure would be a high reward for no collision and a low or negative reward if a collision was experienced. However, if this naïve reward structure were used, the robot would quickly learn that it could achieve the maximum reward by simply doing nothing. The reward structure summarised in Table 7.1 was found to produce more appropriate behaviour, given that a high degree of movement is also desirable. Note that these values were chosen so that repetitive behaviour, such as moving backwards and forwards, will also result in a relatively low reward.

Table 7.1: Reward given for each of the nine possible robot actions. If a collision is experienced the reward is set to zero.

	Turn Left	Do Not Turn	Turn Right
Move Forward	0.8	1.0	0.8
Stop	0.6	0.5	0.6
Move Backward	0.5	0.4	0.5

7.2.2. Exploration and Learning Strategy

For these experiments, actions were selected using the Boltzmann distribution (equation 7.6) to provide a balance between exploration and exploitation (no

annealing policy was used with temperature $\tau = 0.1$ throughout). Therefore, an on-policy reinforcement learning algorithm such as SARSA will not learn a policy that is optimal, but rather the optimal policy taking into account the possibility of exploratory actions. On the other hand, an off-policy method such as Q-learning will learn the optimal policy. However, the non-optimal policy learned by SARSA will be safer, since it will avoid situations where a poorly chosen exploratory action will lead to a collision. Despite this added safety, Q-learning was chosen as the reinforcement learning paradigm for these experiments. The reason for this decision was that, in the final navigational system, the collision avoidance sub-system is not directly responsible for action selection, and hence an on-policy method would not be suitable.

7.2.3. Input Representation

One of the most important decisions to be made when designing a reinforcement learning system is how states will be represented or measured. In this case, the information available to the robot consists of the range estimates of each of the sensors, and the collision indicator. Also available in the completed system, will be output from the view cell and place cell populations. Of these two, view cells seem to be a more appropriate source of input to the collision avoidance system. View cells offer a high degree of generalisation between environments, since similar views should exist. View cell input should also be more naturally correlated with appropriate actions than place cells. In addition, place cell firing is not generally correlated with head direction, and the appropriate action from a given place may vary dramatically with the direction the robot is facing.

Three architectures were implemented and compared. In the first, the state representation consisted of the raw sensor information. From this, a linear value function was learned using the process described in section 7.1.5. The second architecture used the winning view cell as the state for a simple table-based reinforcement learning agent. Thirdly, both techniques were combined, using the linear function approximation technique, but with view cells as input.

7.2.4. Testing

The collision avoidance system was tested using the simulation described in Appendix A. Figure 7.8 shows the performance of the robot for different input representations, while Figure 7.9 shows the training environment and typical paths that were learned.

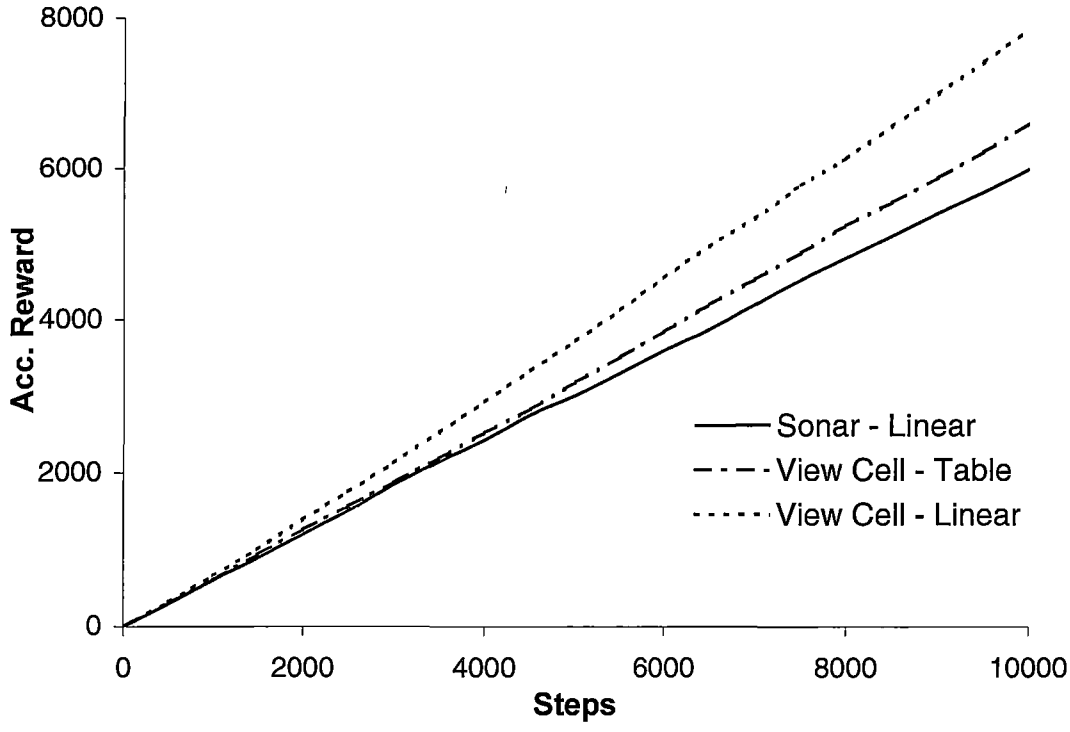

Figure 7.8: Performance of the mobile robot performing the collision avoidance task. Three Q-learning strategies were compared: linear function approximation with raw sensor readings as input, linear function approximation with view cells as input, and table based Q-learning using the winning view cell as the current state. (error bars are insignificantly small and have been omitted)

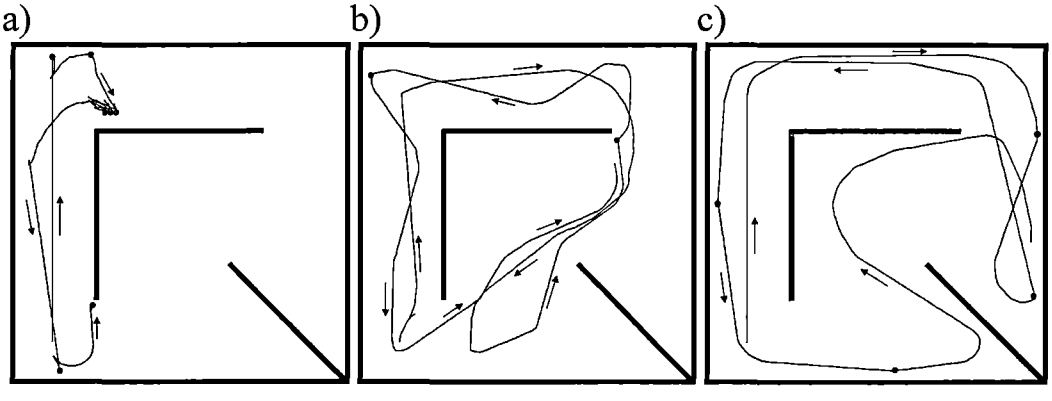

Figure 7.9: Typical paths learned using a) linear approximation from raw sensor data, b) table lookup using the winning view cell as the current state, and c) linear approximation from view cell output.

Figure 7.8 shows that Q-learning was unable to learn the action value function using linear approximation from raw sensor readings. The average reward under this strategy is little better than could be expected for the random selection of actions from Table 7.1, and the path in Figure 7.9a shows that suitable behaviour was not

learned. This is because, at the time scale used, successive sets of sensor readings differ only slightly, and hence it is difficult to apply the TD algorithm. In theory, it should be possible to solve the problem if the parameters λ and γ are carefully selected, however in practice this proved very difficult.

The table-based approach, using the winning view cell as the current state, did result in suitable behaviour, as shown in Figure 7.9b. However, Figure 7.8 shows that the performance is worse than that using linear approximation from view cells. While both of these methods result in few collisions, paths for the tabular approach are more erratic than those for linear approximation (Figure 7.9c). When using tabular reinforcement learning, no generalisation is made between similar views, and therefore, as each new view is experienced, the value function needs to be learned from scratch. In addition, the combined output of several view cells produces a more precise estimate of the current state than can be achieved by considering the winning view cell only.

As expected, linear approximation using view cells as input gave the best performance. The view cells respond to subtle differences in sensor input, enabling better state discrimination than using raw sensor input. Furthermore, the ensemble output of view cells allows greater generalisation, and hence faster learning, than using a tabular approach based on the winning view cell.

7.3. Summary

This chapter has reviewed reinforcement learning techniques from the group of algorithms known as temporal difference learning. On e of these techniques, *Q*-learning was used to develop the low-level navigation system. In particular, it was found that *Q*-learning, using action values computed from a linear combination of view cell outputs, produced good performance for the exploration and collision avoidance task.

Chapter 8. High-Level Navigation

8.1. Goal-Independent Learning

Reinforcement learning (RL) techniques, such as TD learning (Sutton, 1988), have been shown to display good performance in tasks involving navigation to a fixed goal (Foster et al., 2000; Sutton & Barto, 1998). However if the goal location is moved, the previously learned information interferes with the task of finding the new goal location, and performance suffers accordingly (Foster et al., 2000). Since a mobile robot needs to be able to navigate throughout its environment performing tasks at potentially any location, a more flexible learning algorithm is required.

Rats provide us with a good example of an animal with navigational abilities similar to that desirable for a mobile robot. Rats exhibit "one-trial learning" in tasks where the goal location is moved after learning to navigate to a previous location, as shown in the Morris watermaze experiments (see Figure 2.1).

Foster and colleagues (Foster et al., 2000) explored the use of place cells for navigation in the watermaze tasks using TD-learning. It was found that the performance of the actor-critic (Barto et al., 1983) architecture was qualitatively similar to that of a rat when the platform location was not moved. However, as expected, it was not able to achieve one-trial learning when the platform was moved.

To overcome this problem, Foster and colleagues used TD-learning in a novel way to learn a mapping from the place cells to a coordinate system. As the coordinate mapping became more accurate, the system was able to utilise this information to compute direct paths to the goal location. The coordinate learning was goal independent, and could be used to achieve one-trial learning when the platform was moved.

A limitation of Foster and colleagues' method is the inability to deal appropriately with complex environments involving barriers and dead-ends. In such environments, computing the direction to a goal location may not provide any useful information, and may even be counter-productive. In the worst case scenario, this system will revert to using the goal dependent RL technique only, and will not be able to achieve one-trial learning.

Dyna-Q (Sutton, 1990) is a reinforcement algorithm that learns a model of the environment. The model is used to generate simulated experiences, hence allowing

additional value updates. Dyna-Q is able to deal more successfully with dynamic goals, since many simulated experiences may be generated when changes to the model are identified. Many updates based on simulated experiences will enable Dyna-Q to achieve one-trial learning when a goal location changes, or when the environment changes, but these offline updates will require considerable computing resources.

The DG-learning algorithm (Kaelbling, 1993b) is capable of learning in a manner that is completely goal independent, and thus achieves one-trial learning in watermaze tasks. In addition, DG-learning may be applied in complex environments. The algorithm learns the expected number of steps, $DG(s,a,g)$, from each state, *s,* to each other state, *g,* for a given action, *a.* The learning rule is a slight variation of the *Q*-learning update rule:

$$DG(s,a,g) \leftarrow \alpha\left(1+\min_{a'} DG(s',a',g)\right)+(1-\alpha)DG(s,a,g) \qquad \textbf{8.1}$$

where s' is the nest state and $DG(s,a,g)$ is defined to be zero if $s=g$. This update rule is applied for all g at each time steps. Actions are selected by choosing the action that minimises $DG(s,a,g)$ for a given goal.

While the DG-learning algorithm is goal independent, it does not include a method for applying eligibility traces. That is, the update rule relates to the most recent action only. However, it is possible to apply the triangular inequality to update other action values. In terms of DG-learning, the triangular inequality is written:

$$DG(s_1,a,s_2) \le DG(s_1,a,s_i)+\min_{a'} DG(s_i,a',s_2) \qquad \textbf{8.2}$$

This rule may be used to derive additional value updates called relaxations:

$$DG(s_1,a,s_2) \leftarrow \min\left(DG(s_1,a,s_2), DG(s_1,a,s_i)+\min_{a'} DG(s_i,a',s_2)\right) \qquad \textbf{8.3}$$

The Floyd-Warshall algorithm provides a relaxation method that converges on the correct function after $|S|^3$ steps, or $|S|^3 \times |A|$ steps in the case of DG-learning, as shown in Figure 8.1.

$$
\begin{array}{l}
\text{for } i = 1 \text{ to } |S| \\
\quad \text{for } j = 1 \text{ to } |S| \\
\quad\quad \text{for } k = 1 \text{ to } |S| \\
\quad\quad\quad \forall\, a \in A \\
\quad\quad\quad\quad DG(s_j, a, s_k) \leftarrow \min\left(DG(s_j, a, s_k), DG(s_j, a, s_i) + \min_{a'} DG(s_i, a', s_k) \right)
\end{array}
$$

Figure 8.1: The Floyd-Warshall algorithm for DG-learning. Note that the order of iteration is important, the intermediate states (outer loop) should vary most slowly

It would not be practical to run the complete relaxation algorithm after each reinforcement learning update, therefore Kaelbling suggests performing relaxation in parallel to reinforcement updates. Unfortunately, the relaxation procedure can only find shorter paths, and will produce poor performance if novel obstructions are encountered. In other words, the DG algorithm will be able to achieve one-trial learning when a novel goal is located, or when a novel shortcut is found, but will not perform well when a novel obstruction is encountered.

The following section presents a new algorithm that is similar to DG-learning. The new algorithm, based on Q-learning, improves upon the relaxation procedure of DG-learning to achieve much better performance in situations involving novel obstructions.

8.2. Concurrent Q-Learning

To achieve the level of goal independence required, an algorithm that solves the reinforcement learning problem for all possible goal locations (place fields) concurrently was developed. Having learned this map-like representation, it is possible to navigate from any location directly to any other location, whether that location has previously been a goal location or not. The method is similar to DG-learning (Kaelbling, 1993b) but is based more directly on Q-learning (Watkins, 1989; Watkins & Dayan, 1992).

For path finding, the agent should receive a reward only when the designated goal is reached. The concurrent Q-learning algorithm (CQL) maintains an independent set of action values for each state as though that state was the goal, regardless of the actual location of the current goal. The action value for reaching a goal state s^d after performing action a from the current state s is denoted $Q^{s^d}(s,a)$, and we define $Q^s(s,a)=1\ \forall s,a$. The learning rule for CQL is identical to Q-learning, except that

each set of action values is updated by considering each state in turn as the goal location. That is for each state, action values are updated by applying the learning rule with a reward of 1 if the current state is the goal being considered, and 0 otherwise. The basic CQL algorithm is given in Figure 8.2 below (note that $\max_{a'} Q^{s^d}(s', a') = 1$ if $s'=s^d$).

Initialise $Q^{s^d}(s, a)\ \forall s^d, s \in S,\ \forall a \in A,$
Initialise s
Repeat:
    Choose a from s
    Take action a; observe s'
    For each state (destination), $s^d \in S$:
        $\delta \leftarrow \gamma \max_{a'} Q^{s^d}(s', a') - Q^{s^d}(s, a)$
        $Q^{s^d}(s, a) \leftarrow Q^{s^d}(s, a) + \eta\delta$
    $s \leftarrow s'$

Figure 8.2: The Concurrent Q-Learning algorithm.

Having learned a 'map' of the environment, all that remains is to choose an appropriate action. That is, a state must be chosen as an ultimate goal and an action must be chosen to move towards this goal. To do this the agent must first have an estimate of the expected reward, r^s, for each state s. Given the reward vector, the expected return, $R^s(s',a)$, of moving towards state, s, via an action, a, from the current state s', can be calculated. The state-action pair with the highest expected return is then chosen as the current goal and action, as shown in Figure 8.3.

For all s^d, a
    $R^{s^d}(s, a) \leftarrow Q^{s^d}(s, a) \times r^{s^d}$
$s^T \leftarrow \arg_{s^d} \max_{s^d, a} R^{s^d}(s, a)$
$a^T \leftarrow \arg\max_{a} R^{s^T}(s, a)$

Figure 8.3: Greedy action selection algorithm for the CQL algorithm. s is the current state, s^T is the selected target state, and a^T is the selected action.

This greedy action selection algorithm may easily be modified to use an ε-greedy strategy (as in the current work) or similar non-greedy strategy.

As with DG-learning, relaxation may be used to improve performance. Relaxation is the process of enforcing the triangular inequality:

$$\overline{AC} \leq \overline{AB} + \overline{BC} \qquad \textbf{8.4}$$

This equation can be converted to the action value representation:

$$Q^C(A,a) \geq Q^B(A,a) \times \max_{a'} Q^C(B,a') \qquad \textbf{8.5}$$

This rule would ideally be applied to all possible state and action combinations for each iteration of the learning algorithm. However, the complexity of the full relaxation procedure, $O(|S|^3 \times |A|)$, would make this impractical. Therefore for CQL, relaxation is only performed for paths involving the most recently experienced state transition, thereby reducing the complexity to $O(|S|^2 \times |A|)$. The CQL algorithm, including relaxation, is given in Figure 8.4.

Initialise $Q^{s^d}(s,a)\ \forall s^d, s \in S,\ \forall a \in A,$
Initialise s
Repeat:
  Choose a from s
  Take action a; observe s'
  For each state (destination), $s^d \in S$:
    $\delta \leftarrow \gamma \max_{a'} Q^{s^d}(s',a') - Q^{s^d}(s,a)$
    $Q^{s^d}(s,a) \leftarrow Q^{s^d}(s,a) + \eta\delta$

  Relaxation
  for all state-action pairs (s^o, a^o)
    $\delta^o \leftarrow Q^s(s^o,a^o)\gamma \max Q^{s^d}(s',a) - Q^{s^d}(s^o,a^o)$
    if $\delta^o > 0$
      $Q^{s^d}(s^o,a^o) \leftarrow Q^{s^d}(s^o,a^o) + \eta\delta^o$
  $s \leftarrow s'$

Figure 8.4: The Concurrent Q-Learning algorithm with relaxation.

Unfortunately, relaxation can only be used to find shorter paths. Therefore, to make the most effective use of relaxation, action values should be initialised with pessimistic values, and this can have a negative impact on exploration (Kaelbling, 1993b). This effect on exploration can be addressed by assigning an exploration bonus to either the states, or state-action pairs.

A state-based exploration strategy may be implemented by direct modification of the reward vector prior to action selection. To do this each location is assigned an initial estimate, r_{eq}, of the expected reward for that state. At each time step, the reward for the current location is updated to the experienced reward (in the watermaze this would be 1 if the platform is reached, and 0 otherwise). The value for all other locations decay towards r_{eq} at some small rate. Since this strategy only modifies the reward vector, it does not alter the effectiveness of the relaxation procedure. Therefore, action values may be pessimistically initialised without adversely affecting initial exploration. In the extreme case, r_{eq} may be set to 1. Note that these rewards are used for action selection only, action values are still updated independently of the current goal or experienced rewards.

In some situations, exploring each state may not be sufficient to find new paths, and instead, the exploration of each action from each state must be encouraged. In those cases where a state-based exploration strategy is not sufficient, a system similar to that used in the Dyna-Q+ algorithm (Sutton & Barto, 1998) can be implemented.

In the Dyna-Q+ algorithm, state-action pairs are assigned an additional reward, based on the time since this action was last performed. This additional reward is used for training in both the learning and planning stages of the algorithm. Such a system would have an undesirable effect on the consistency of action values. However, if this additional reward is used for action selection only, it should still produce the desired effect. In accordance with Sutton and Barto (1998), the CQL algorithm will be denoted by a '+' when this strategy is used. If $n(s,a)$ is the number of time steps since the state-action pair was visited, and κ is some small number, then the action selection algorithm can be modified to incorporate this exploration strategy as shown in Figure 8.5.

For all s^d, a

$$R^{s^d}(s,a) \leftarrow \left(Q^{s^d}(s,a) + \kappa\sqrt{n(s,a)}\right) \times r^{s^d}$$

$$s^T \leftarrow \arg_{s^d} \max_{s^d,a} R^{s^d}(s,a)$$

$$a^T \leftarrow \arg\max_a R^{s^T}(s,a)$$

Figure 8.5: Action selection algorithm for CQL+ (the CQL algorithm with bonus given for exploratory actions). s is the current state, s^T is the selected target state, and a^T is the selected action. $n(s,a)$ is the number of time steps since action a was chosen from state s, and κ is some small constant.

These exploration strategies enable pessimistic initialisation of action values as required for efficient use of the relaxation procedure. However, another disadvantage of the relaxation process is that, if a novel obstruction is encountered, relaxation will not be able to update all action values for paths that are now closed. This problem is addressed in sections 8.2.1 and 8.2.2.

8.2.1. Adding Eligibility Traces to CQL

As with other forms of TD-learning, the learning rate of CQL may be significantly improved if eligibility traces are included. Two methods of implementing eligibility traces for Q-learning are Watkins' $Q(\lambda)$ (Watkins, 1989; Watkins & Dayan, 1992) and Peng's $Q(\lambda)$ (Peng & Williams, 1996). The method most appropriate for CQL is Watkins' $Q(\lambda)$, since Peng's Q(λ) is not truly an off-policy method (Sutton & Barto, 1998, p184).

The implementation of Watkins' $Q(\lambda)$ for CQL requires an eligibility trace to be kept for each goal state. All traces for a state-action pair are set to 1 whenever that action is chosen. Whenever a non-optimal action is chosen, with respect to a particular goal, that goal's eligibility trace is set to zero for all state-action pairs. The eligibility trace for the current location is also set to zero for all state-action pairs. The new algorithm, denoted CQL-e, is given in Figure 8.6.

Initialise $Q^{s^d}(s,a), e^{s^d}(s,a)\ \forall s^d, s \in S,\ \forall a \in A,$

Initialise s, a

Repeat:

Take action a; observe s'

Choose a' from s'

For each state (destination), $s^d \in S$:

$e^{s^d}(s,a) \leftarrow 1$

$a^* \leftarrow \arg\max_b Q^{s^d}(s',b)$

$\delta \leftarrow \gamma Q^{s^d}(s',a^*) - Q^{s^d}(s,a)$

For all state-action pairs (s^o, a^o):

if $e^{s^d}(s^o,a^o) > 0$:

$Q^{s^d}(s^o,a^o) \leftarrow Q^{s^d}(s^o,a^o) + \eta\delta\, e^{s^d}(s^o,a^o)$

else:

Relaxation

$\delta^o \leftarrow Q^s(s^o,a^o)\gamma \max_a Q^{s^d}(s',a) - Q^{s^d}(s^o,a^o)$

if $\delta^o > 0$

$Q^{s^d}(s^o,a^o) \leftarrow Q^{s^d}(s^o,a^o) + \alpha\delta^o$

Trace Update

if $a' = a^*$ and $s' \neq s^d$

$e^{s^d}(s^o,a^o) \leftarrow \lambda\gamma\, e^{s^d}(s^o,a^o)$

else

$e^{s^d}(s^o,a^o) \leftarrow 0$

$s \leftarrow s'; a \leftarrow a'$

Figure 8.6: The Concurrent Q-Learning algorithm with relaxation and eligibility traces (CQL-e).

Eligibility traces will allow values to be updated in some situations where no update is possible through relaxation. For example, if a novel obstruction is encountered, then all state-action pairs leading to the novel experience will have their action values updated. However in certain circumstances, this may be only a small subset of the action values potentially affected by this obstruction. The next section introduces a further modification of the CQL algorithm that updates all values affected by each experience.

8.2.2. Using Q-Values for More Efficient Learning

The relaxation method allows all relevant action values to be updated if a novel shorter path is found. However, it is not able to make any updates when a novel obstruction is encountered. In Figure 8.7 CQL would update only the action value E-F. CQL-e is able to update more action values in this situation. If CQL-e was used, the action values B-C, C-D, D-E as well as E-F would be updated (A-B would not be updated since this action is not optimal with respect to G). However, even these additional updates are far less than are conceptually possible, as shown below.

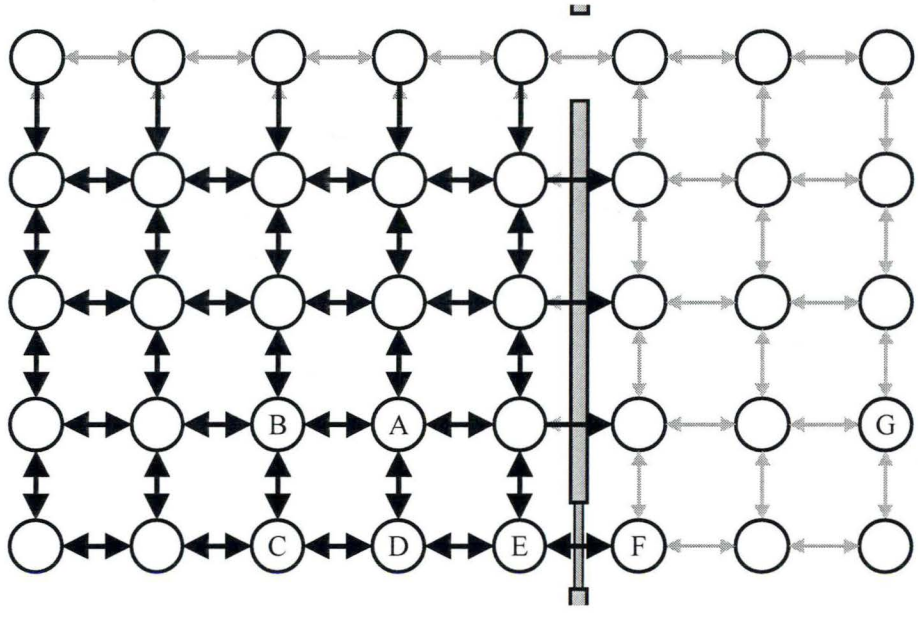

Figure 8.7: A navigational problem consisting of a grid of states with possible actions to each adjacent state; a wall with two 'doorways' divides the environment into two regions. An agent familiar with the environment has just moved along the path A-B-C-D-E and attempts to move to F, but the doorway, previously open, has been blocked. Clearly, the value of taking the action E→F, with respect to the goal G, should be reduced. However in addition, any action for which the subsequent optimal path to G previously included the action E→F should have its value reduced. The actions for which this is the case are identified in bold.

In order to perform all possible updates in Figure 8.7, we need a method for determining that an action is on an optimal path to a given goal. If (s,a) is the state action pair for which an error has just been observed, and if:

$$Q^{s^d}(s^o, a^o) \leq Q^s(s^o, a^o) \times Q^{s^d}(s, a) \qquad \textbf{8.6}$$

then the state-action pair (s,a) must be on the shortest path from s^o, via a^o, to s^d. Therefore, any error observed in the action value for (s,a) must also be applicable to

(s^o,a^o). To make the appropriate update we can replace the eligibility trace $e^{s^d}(s^o, a^o)$ with the corresponding action value, and apply the update rule whenever equation 8.6 holds, as shown in the new algorithm denoted CQL-q. The modified algorithm is shown in Figure 8.8.

Initialise $Q^{s^d}(s,a)\ \forall s^d, s \in S,\ \forall a \in A,$
Initialise s, a
Repeat:
Take action a; observe s'
Relaxation
For each state (destination), $s^d \in S$:
For each action a' and intermediate state, $s' \in S$:
if $Q^s(s',a') \times \max_a Q^{s^d}(s',a) > Q^{s^d}(s',a')$
$Q^{s^d}(s',a') \leftarrow Q^s(s',a') \times \max_a Q^{s^d}(s',a)$
Choose a' from s'
For each state (destination), $s^d \in S$:
$\delta \leftarrow \gamma \max_b Q(s',b) - Q^{s^d}(s,a)$
For all state-action pairs (s^o, a^o):
if $Q^s(s^o,a^o) \times Q^{s^d}(s,a) \geq Q^{s^d}(s^o,a^o)$
$Q^{s^d}(s^o,a^o) \leftarrow Q^s(s^o,a^o)\left[Q^{s^d}(s,a) + \eta\delta\right]$
$s \leftarrow s'; a \leftarrow a'$

Figure 8.8: The Concurrent Q-Learning algorithm with relaxation and 'Q' updates (CQL-q).

While the new update rule largely eliminates the need for relaxation, in certain situations CQL-q may make value updates that are unduly pessimistic. For example, if two independent paths of equal length exist to a given goal, then if one path becomes blocked, the values for actions leading to the origin of both paths will be reduced erroneously. Therefore, relaxation is still required to correct action values that may have been trained too low. However, relaxation now only needs to be performed for the current state, prior to action selection.

8.2.3. CQL Performance in the Watermaze

To test the ability of the algorithm to deal with dynamic environments, CQL was applied to the RMW and DMP tasks as described in Steele and Morris (1999). Input to the learning system was via 400 simulated place cells, which is comparable to the number of place cells in Foster and colleagues (2000). Unlike Foster and colleagues however, the environment was discretised into 400 corresponding locations in a square 20×20 grid (note that some locations are not accessible). Movement was restricted to the eight adjacent locations, with the action being performed in one time step. The platform was the same size as a single place field, making it proportionately the same size as in Steele and Morris (1999). Platform locations were chosen to minimise the possibility of straight-line movement between platform and start locations. The environmental setup is shown diagrammatically in Figure 8.9.

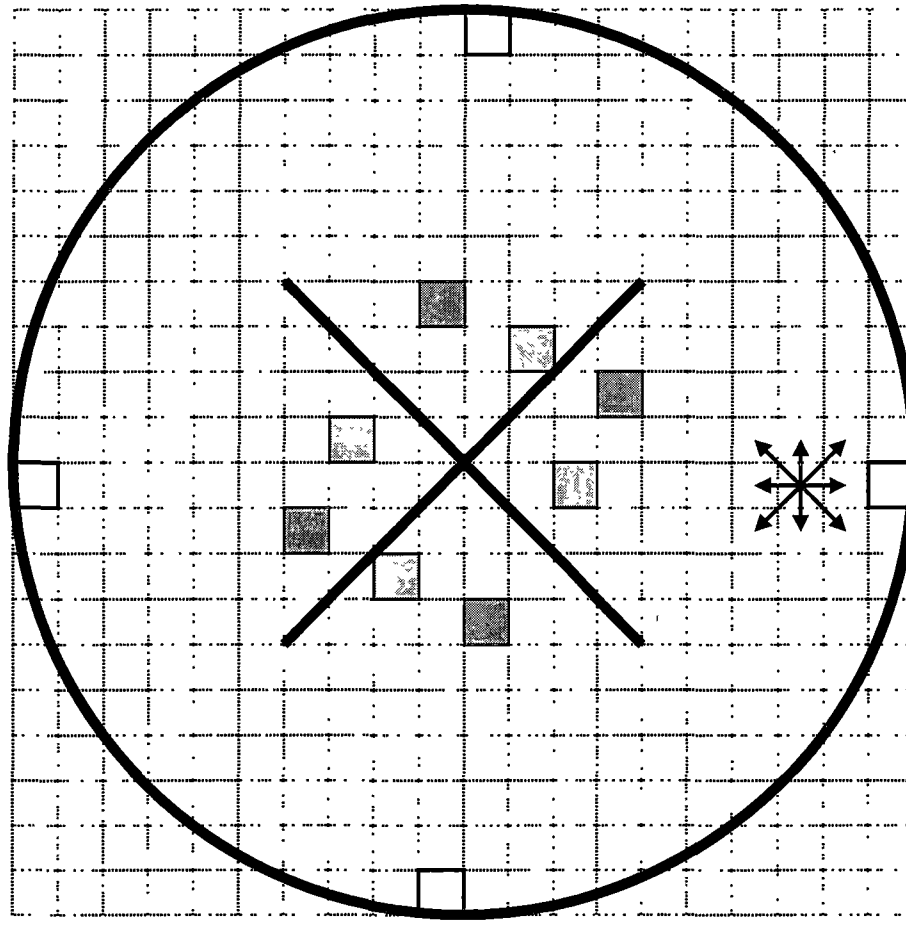

Figure 8.9: The environment used for the watermaze task showing: place cell locations (grid), start locations (white squares with bold outline), and goal locations (solid squares). The dark grey squares were used as goal locations for the RMW task. All solid squares were used as possible goal locations for the DMP task. The central barrier is also shown for tasks where this was included.

For the watermaze task, each location was assigned an initial estimate, $r_{eq,}$ of the expected reward for that state. At each time step, the reward for the current location was updated to the experienced reward (that is: 1 if the platform is reached, and 0 otherwise). The value for all other locations decayed towards r_{eq} at some small rate. In combination with the action selection algorithm (Figure 8.3), this process achieves a good balance between exploration and exploitation. In addition, to improve the efficiency of searching when the platform location is unknown, the system was given a slight preference for travelling in the same direction as that chosen at the previous time step. This is similar to Foster and colleagues' decision to add momentum to their system. In addition, non-greedy actions were limited to the two directions adjacent to the greedy action.

Figure 8.10 compares the performance of Q(λ), CQL-e, CQL-q and DG-learning in the RMW and DMP tasks (variance is detailed in Table 8.1). Like the actor-critic architecture in Foster and colleagues (2000), $Q(\lambda)$ was able to learn the initial goal location quite quickly. However as expected, performance suffered when the goal location changed. This was particularly noticeable in the DMP task, where repeated interference from previous platform locations caused a progressive degradation in performance. The improved performance over the actor-critic architecture in Foster and colleagues (2000) was primarily due to the inclusion of eligibility traces.

The CQL-e algorithm learned faster than $Q(\lambda)$ and also showed very good one-trial learning when the platform location changed in the RMW task. In the DMP task, CQL-e was able to achieve one-trial learning by day four to five. DG-learning performed slightly better than CQL-e, presumably this was due to the alternate cost-per-step representation used. CQL-q learned faster than all other algorithms, due to the increased number of updates made at each time step.

To confirm the ability of CQL to handle complex environments, the algorithm was also tested in a watermaze environment with a centrally located barrier as shown in Figure 8.9. Due to time constraints, only CQL-e was tested. The performance difference between the RMW and DMP tasks, with and without barriers, is no greater than would be expected given the greater path lengths required when the barrier is present. This confirms the ability of CQL-e to handle complex environments. CQL-q and DG-learning were later tested in similarly complex environments, as shown below in Figure 8.12, and both would be expected to perform well in the watermaze task with barriers.

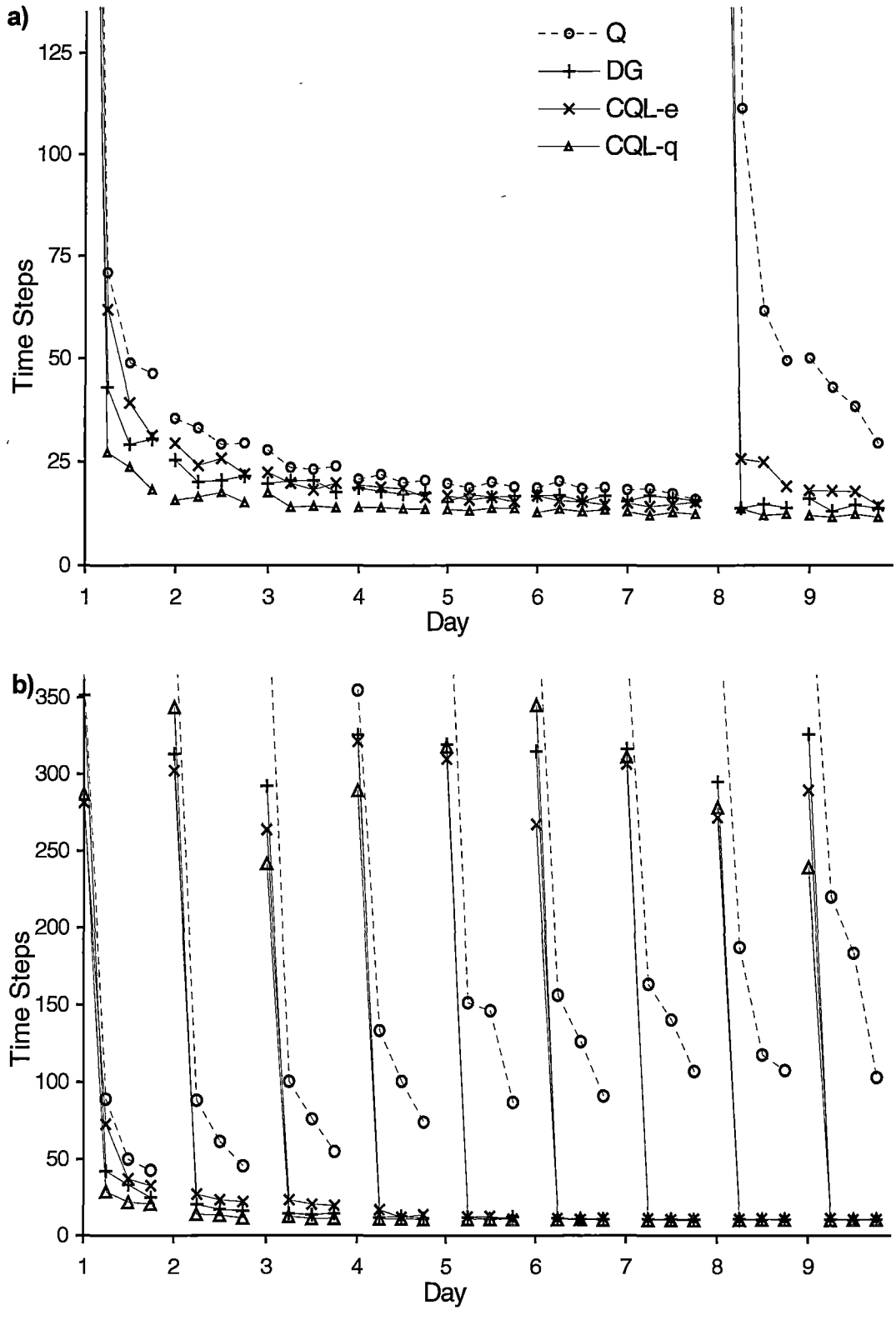

Figure 8.10: Performance of Watkins' $Q(\lambda)$, Concurrent Q-learning and DG-learning on the RMW (a), and DMP (b) tasks. λ=.95 for Watkins' $Q(\lambda)$ and CQL-e, γ=.90 in all cases. Error bars have been omitted for clarity.

Table 8.1 Mean time to find the platform on the fourth trial of each 'day', for each algorithm, including 95% confidence intervals.

RMW	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9
Q- Learning	46±9	29±5	24±4	21±2	19±2	19±2	16±2	49±13	29±6
DG-Learning	31±5	21±3	18±2	18±2	17±2	17±2	16±2	14±2	14±2
CQL-e	31±5	22±3	20±2	17±2	15±2	15±2	15±2	19±4	14±2
CQL-q	18±3	15±2	14±1	14±1	14±1	13±1	12±1	12±1	12±1

DMP	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9
Q- Learning	43±9	45±9	55±10	74±16	87±19	91±19	107±27	107±25	102±22
DG-Learning	25±4	16±2	14±2	11±1	12±1	11±1	10±1	10±1	11±1
CQL-e	33±5	22±4	20±4	14±2	11±1	11±1	11±1	10±2	10±1
CQL-q	21±3	11±1	11±1	11±1	11±1	10±1	10±1	10±1	10±1

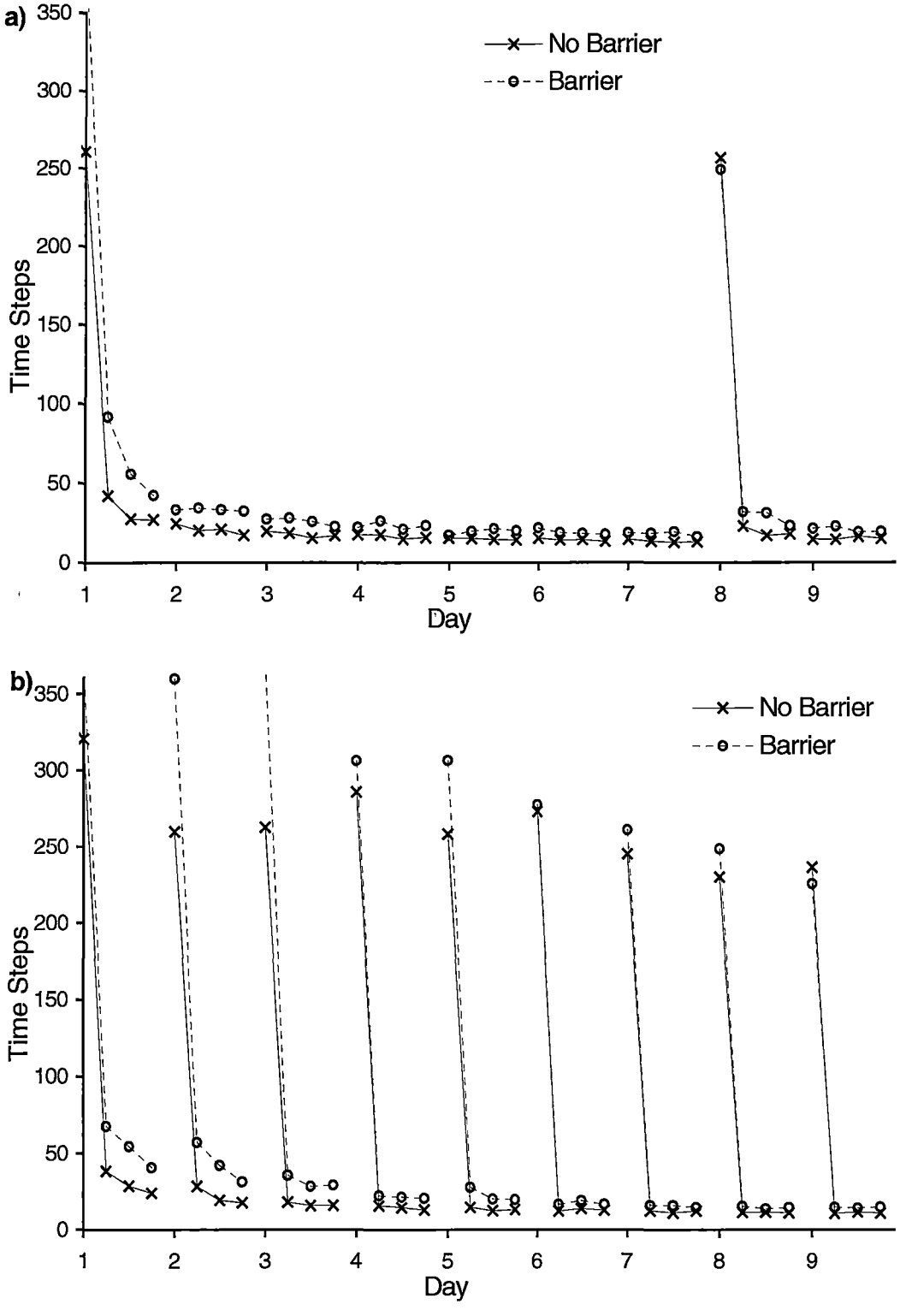

Figure 8.11: Performance of CQL-e on the RMW (a) and DMP (b) tasks in watermazes with and without a centrally located barrier. λ=.95 in all cases, γ=.90 for trials with no barrier and γ=.95 in trials with barrier. Error bars have been omitted for clarity.

8.2.4. CQL Performance in Dynamic Environments

To test the performance of the CQL algorithm in changing environments, a grid-world task was devised similar to those used by Sutton and Barto (1998). A 10x10 square environment was divided into two equal sized "rooms". The two rooms were connected by two doors, which could be opened or closed. The goal location alternated between random locations in each room so that the agent had to navigate through the doors for each episode. Movement was allowed only in the North, South, East and West directions.

The detour experiment started with both doors open. After the task had been learned by the agent, one door was closed. The blocking experiment started with one door open. After the task had been learned, this door was closed and the other door opened. The shortcut experiment started with one door open. After a period of time, the other door was opened to create a potential shortcut. Since these tasks require goal independent learning, conventional reinforcement learning techniques will fail. Therefore, the three experiments were conducted for DG-learning, CQL-e and CQL-q, only.

For these tasks, the goal location was given directly to the agent; that is in the action selection algorithm (Figure 8.3), r^s is equal to 1 if s is the goal location and 0 otherwise. This means that the process used in the watermaze tasks to encourage exploration may not be applied here. Therefore, to encourage exploration in these tasks, the action selection algorithm including an exploratory bonus was used (Figure 8.5). In accordance with Sutton and Barto (1998), the CQL algorithms will be denoted CQL-q+ and CQL-e+ when this strategy is used, and DG+ when used with DG-learning.

The environments and results for these three experiments are shown in Figure 8.12.

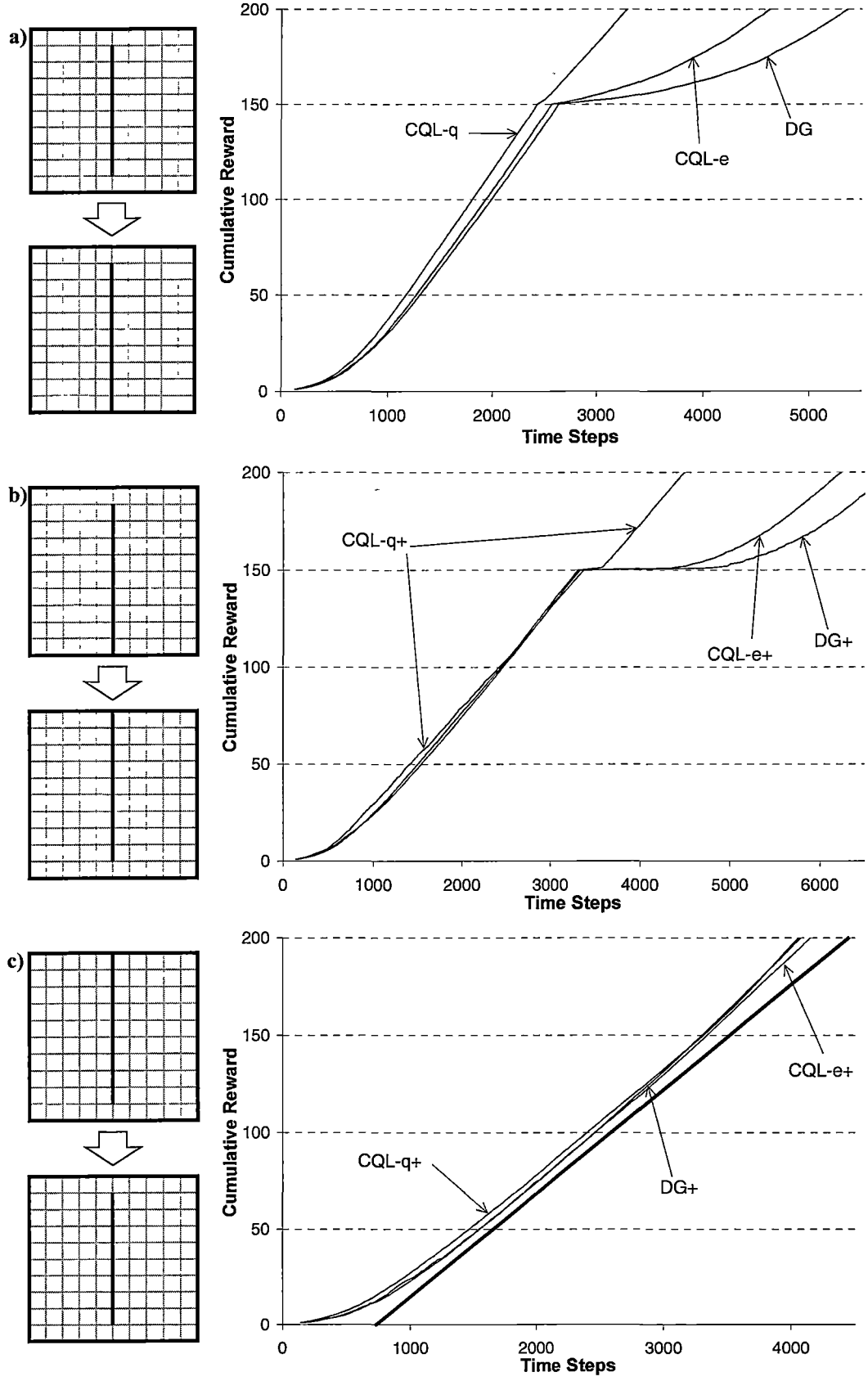

Figure 8.12: Environment setup and results for the a) detour, b) blocking and c) shortcut experiments. The straight bold line in c) is a visual guide only (Error bars for these graphs are insignificantly small and have been omitted).

Figure 8.12a shows that CQL-q was able to solve the detour task more efficiently than both DG-learning and CQL-e. Most importantly, when the door was closed, CQL-q was able to adjust to the new environment very quickly compared to DG-learning and CQL-e, both of which continued to try using the South door long after it was closed.

With DG-learning, relaxation updates are not applicable in this case, as they can not be used to increase the estimated distance of the path via the South door. Hence, much exploration is required to correct previously learned values. As expected, the eligibility trace updates of CQL-e offered some improvement. A more dramatic improvement is seen with the CQL-q algorithm, which is able to update all relevant action values as soon as the closed door is found. These pessimistic updates are then corrected as the agent searches for a shorter path via the relaxation updates.

For the blocking and shortcut tasks, the CQL-q+, CQL-e+ and DG+ algorithms were used. Without the exploratory bonus given during the action selection phase of these algorithms, the CQL and DG-learning algorithms find these problems difficult to solve, as do goal-dependent algorithms, such as Dyna-Q (Sutton & Barto, 1998).

Figure 8.12b shows similar performance for all algorithms before the door states changed. When the doors were switched however, CQL-q+ was able to locate and utilize the new path much more quickly than CQL-e+ or DG+. When the door is closed, CQL-q+ quickly updates all values close to zero. With low action values, the exploration bonus dominates, and an exploratory sweep of the room commences. Once the new door is discovered, CQL-q+ immediately updates all relevant action values, and almost immediately returns to optimal behaviour. In contrast, action values are updated much more slowly by CQL-e+, and even more slowly by DG+. Consequently the exploration bonus is not able to dominate the now incorrect action values, and the agent begins a period of erratic behaviour.

The shortcut task in Figure 8.12c does not involve a novel obstruction, and therefore all algorithms show similar performance in this task. The slight upward curvature of the graphs in the latter half of the experiment indicates that all three algorithms correctly learn to exploit the shortcut.

8.2.5. Hierarchical Learning for Reducing the Complexity of CQL

While Q-learning and, by extension, CQL are efficient learning algorithms in terms of the number of time-steps taken to learn optimal solutions (Koenig & Simmons, 1993), they both suffer the 'curse of dimensionality' with respect to the update time

per step. Given a state space S, and action space A, the worst case update time complexity for Watkins' $Q(\lambda)$ is $O(|S|\times|A|)$, while for CQL it is $O(|S|^2\times|A|)$. Lazy learning may be applied to $Q(\lambda)$ to reduce the complexity to $O(|A|)$ as in the Fast Online $Q(\lambda)$ algorithm (Wierling & Schmidhuber, 1998). However, such techniques would be more difficult to apply to CQL, and at best would only reduce the complexity to $O(|S|\times|A|)$.

One common approach to this type of problem is to employ a tree data structure. Hirtle and Jonides (1985) found that human subjects organised spatial landmarks in a hierarchical manner, and it seems likely that other information may also benefit from being organised in this way. If a tree of states were used in conjunction with CQL, each state would need to learn action values for each of its siblings, each of its parent's siblings, each of its parent's parent's siblings and so on, as shown in Figure 8.13. Additionally, all of a state's siblings would have their action values updated whenever an action is performed from that state.

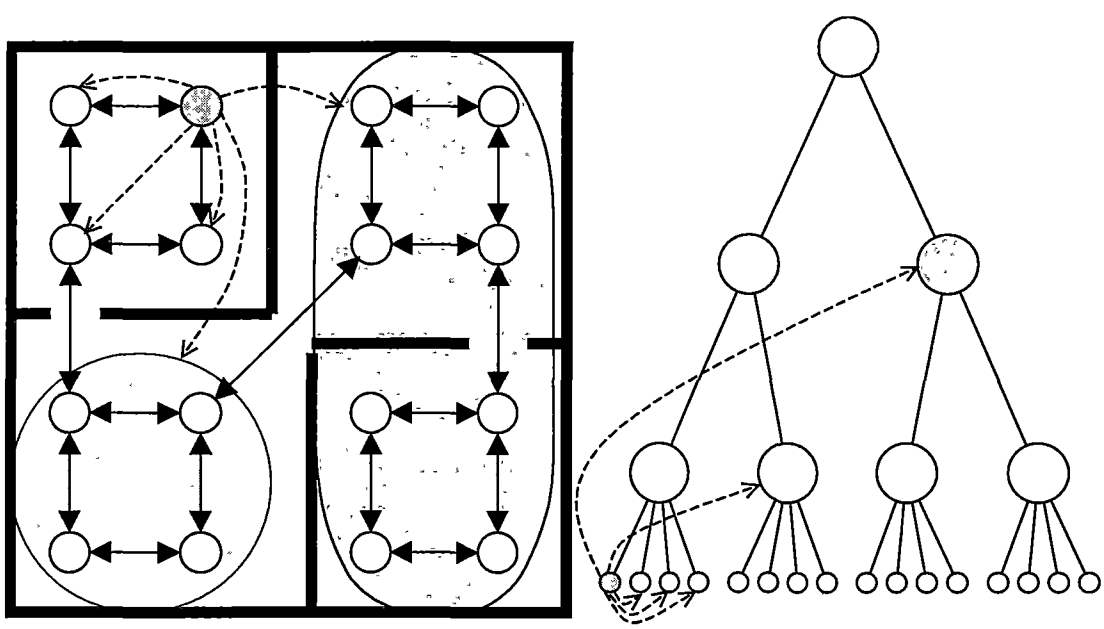

Figure 8.13: Left: A group of states that may be best represented as hierarchical groups. Solid arrows show possible actions, dotted lines show action values that would need to be learned from the shaded state. The shaded ovals show some conceptual groups. Right: A tree structure representing the environment on the left. The leaf nodes of the tree represent the states themselves; other nodes represent a conceptual grouping of the states. Dotted arrows show the action values, corresponding to the diagram on the left, that must be learned from the shaded state. Shaded circles correspond to the conceptual groups in the left diagram.

Previous work has considered similar hierarchical structures to reduce the complexity of reinforcement learning algorithms (Dayan & Hinton, 1993; Dietterich, 1998; Kaelbling, 1993a; Parr & Russell, 1997; Singh, 1992). For the CQL algorithm operating on $|S|$ states arranged in a balanced tree structure of degree d, this approach would yield a theoretical worst case update time per action of:

$$W(|S|) = \left[(d-1) \times \log_d |S|\right]^2 = O(\log |S|) \qquad \textbf{8.7}$$

While this is a significant improvement, there are likely to be few real world problems for which a suitable tree structure can be determined prior to training. For all other problems, the tree structure would need to be determined dynamically. While Digney (1996) presents an algorithm that learns such a structure dynamically, it is not clear that such a technique could easily be applied to CQL.

An alternate, though similar, approach would be to identify some states as being more important than others. These key states would be similar to the parent nodes in the tree structure, with states learning action values to other states based on their proximity, and the degree of importance placed upon them; ignoring states of lesser importance. An algorithm for this truncated form of CQL is given in Figure 8.14.

Initialise $Q^{s^d}(s,a)\ \forall s^d, s \in S,\ \forall a \in A,$
Initialise s, a
Repeat:
  Take action a; observe s'
  Choose a' from s'
  For each state $s^d \in S \mid Q^{s^d}(s',a') > T(s^d)$:
    $\delta \leftarrow \gamma \max_b Q(s',b) - Q^{s^d}(s,a)$
    For each $s^o \in S \mid \max_b Q^{s^d}(s^o,b) > T(s)$:
      For each $a^o \in A$
        if $Q^s(s^o,a^o) \times Q^{s^d}(s,a) \geq Q^{s^d}(s^o,a^o)$
          $Q^{s^d}(s^o,a^o) \leftarrow Q^s(s^o,a^o)\left[Q^{s^d}(s,a) + \eta\delta\right]$
  $s \leftarrow s'; a \leftarrow a'$

Figure 8.14: The truncated CQL algorithm (T-CQL). $T(s)$ is the training threshold assigned to state s. Low thresholds can be considered to represent high importance or key states.

Note that the value update algorithm for T-CQL omits the relaxation step of CQL. It was found that these updates were not necessarily valid in the truncated algorithm. The role of these updates, primarily one of finding shorter paths, has been transferred to the action selection algorithm, which is discussed below.

The update procedure in Figure 8.14 is not sufficient to solve the problem, since states with high thresholds (low importance) will never learn action values to distant states with similarly high thresholds. In order to choose an action that will lead to such a state, the agent needs to search for an intermediate key state with a low threshold that has legitimate action values for the target. The agent then begins to move in the general direction of the target by first moving towards the closest of these intermediate states. As it moves towards the nearest intermediate state, a new state closer to the target may become known, and the trajectory changes towards the new state, as shown in Figure 8.15.

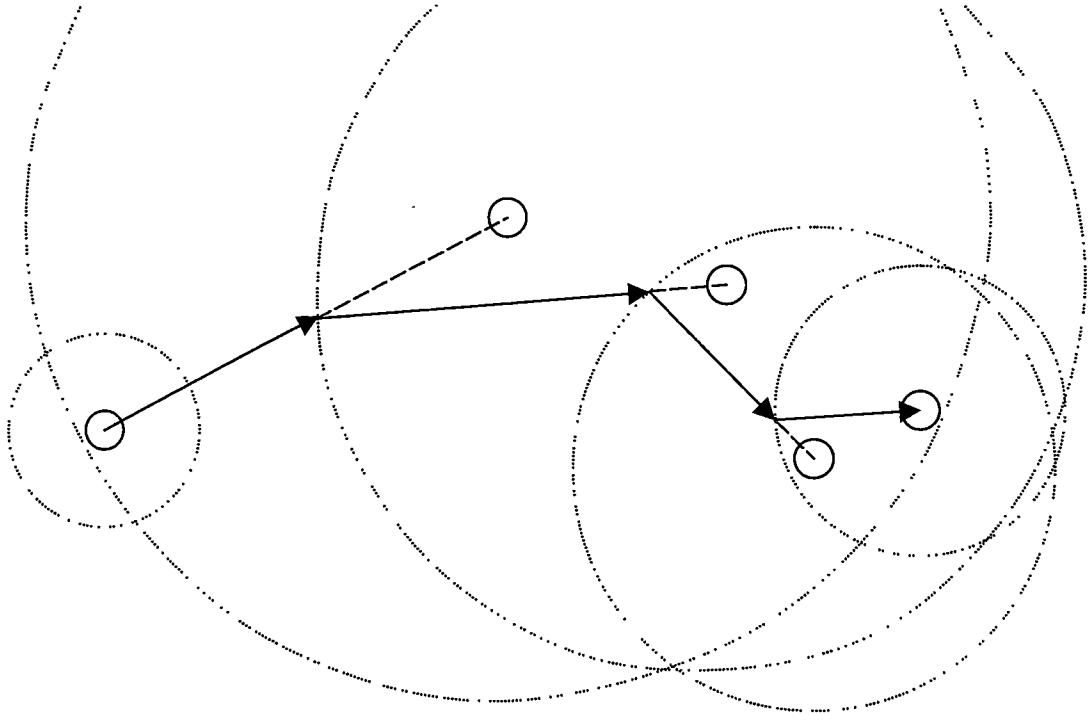

Figure 8.15: A typical trajectory generated using the T-CQL algorithm. Solid circles represent states; corresponding dotted circles represent their training thresholds. Dashed lines show the planned path; solid arrows show the actual path taken.

While Figure 8.15 shows that non-optimal paths may be generated, in practice optimal paths are often found, either through the use of redundancy in choosing key states, or simply because there are a finite number of actions that may be chosen

from any state. The action selection algorithm, as depicted in Figure 8.15, is given below in Figure 8.16.

$$s^t \leftarrow \arg\max_x \left[\max_b Q^x(s,b) \times \max_b Q^{s^t}(x,b) \mid x \in S, \max_b Q^x(s,b) > T(x) \right]$$

$$a \leftarrow \arg\max_b Q^{s^t}(s,b)$$

Figure 8.16: The action selection algorithm for the T-CQL algorithm. s^t is the target state, a is the action that will be performed from the current state, s.

This action selection strategy will be sufficient, provided the agent has explored enough to find suitable key states with low thresholds. In the early stages of training, this may not be the case, and problems may arise. For example, the agent may reach a key state with accurate action values to the current target; the subsequent state may not have learned about the target, the agent then searches for a suitable intermediate state. At this point, the action values of the previous key-state would not normally be updated, because the current state has no information about the target. If little exploration has been undertaken at this point, the agent may find that the most suitable intermediate state is the key state just visited. The agent will then return to the key state and continue back and forth between the two states indefinitely.

To solve this problem, action values are updated for each state that meets the threshold criterion, and for the current target. This may lead to a certain degree of 'forgetting' in the early stages of training since, as in the example above, a key state may have its action values erroneously updated based on incorrect information from the subsequent state. The advantage, however, is that it encourages exploration by forcing the agent to choose an alternate route from the key state.

The final issue is the choice of thresholds or key states. While this may be easier than finding a tree structure for the states, it may still be difficult or impossible to identify key states prior to training. However, it was found that, provided a reasonably conservative distribution function was chosen, the thresholds could be assigned randomly.

In keeping with the tree-like nature of the algorithm, thresholds were chosen from an exponential distribution as shown in equation 8.8:

$$f(x) = \frac{ae^{ax}}{e^{a} - 1} \qquad \textbf{8.8}$$

where a is a parameter controlling the shape of the function.

Distributions for various values of a are shown in Figure 8.17.

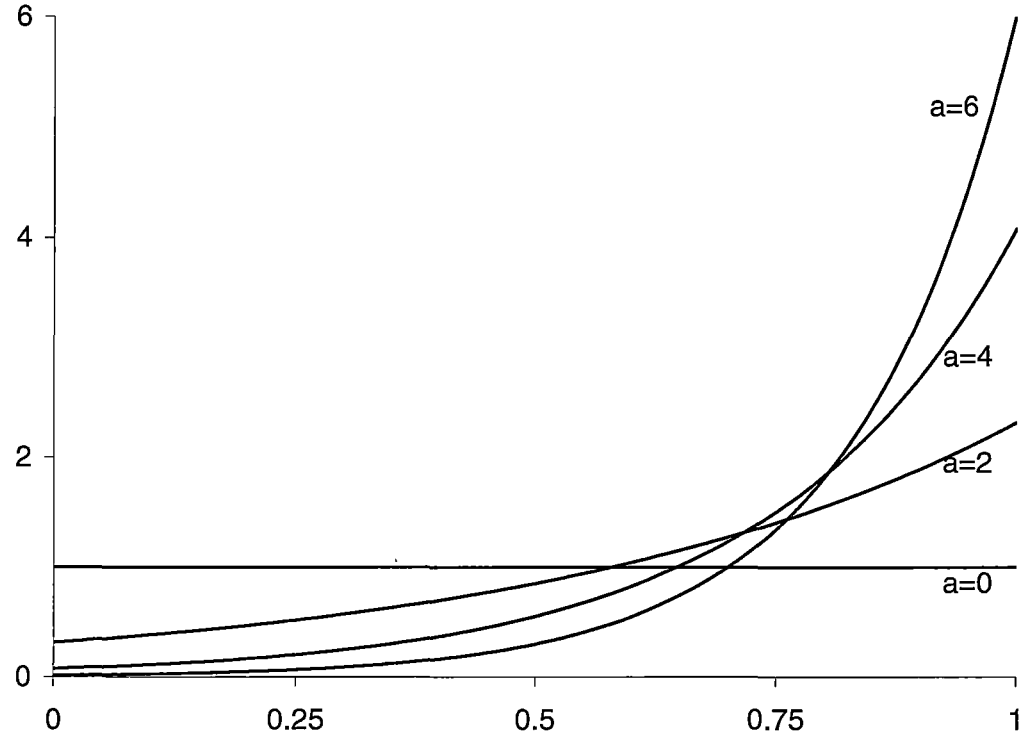

Figure 8.17: Exponential distributions generated for various values of a in equation 8.8.

The worst case time complexity, for both the update and action selection algorithms, will occur when training is near completion, since action values start at zero and more updates are performed when more action values are higher than thresholds. It will also occur for a state near the conceptual centre of the environment, since this state should be within the threshold of a larger number of states than a state at the edge of the environment.

To derive an expression for the update time complexity, we will consider a simple environment consisting of states arranged in a two-dimensional plane. Each state has neighbours to the north, south, east and west, with no barriers. The number of

states that are r steps from the central state is $4r$. Therefore the number of states, $N(R)$, which are within R units of the central state, and which need to be considered in the update algorithm is:

$$N(R) = \sum_{r=1}^{R} 4r\, \mathcal{P}(T < \gamma^r) \qquad \textbf{8.9}$$

For any threshold probability distribution that it not asymptotic at r=0, this is a convergent series. For example, if the thresholds are distributed evenly between 0 and 1, then in the limit as $R \to \infty$:

$$N(R) = \sum_{r=1}^{R} 4r\gamma^r \simeq \frac{4\gamma}{(1-\gamma)^2} \qquad \textbf{8.10}$$

This may easily be extended to the general case, giving worst-case time complexities for the update and action selection algorithms of $O(|A|)$, provided the probability distribution is not asymptotic at zero.

The T-CQL algorithm was tested in a complex office-like environment consisting of 256 states, as shown in Figure 8.18. Possible actions consisted of the four compass points: north, south, east and west. The agent was required to navigate between successive random or pseudo-random locations within the environment. The successful traversal from one location to another constituted one episode.

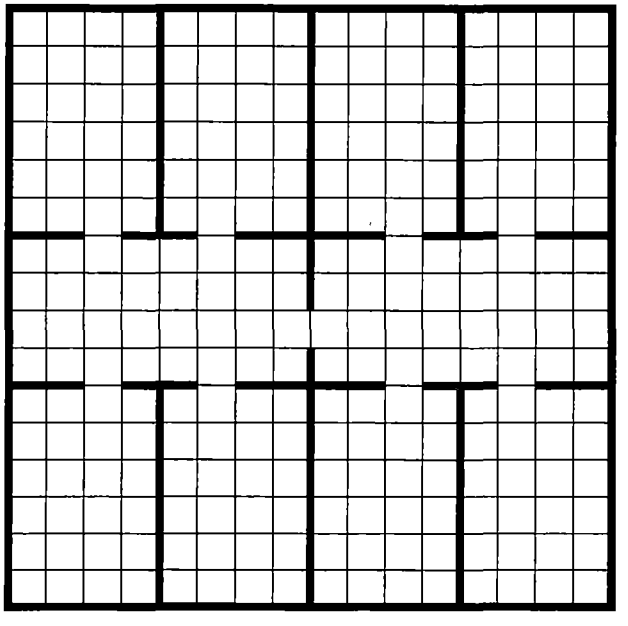

Figure 8.18: The environment used for testing T-CQL. Thick lines represent walls; thin lines represent state divisions.

Threshold values were chosen from random exponential distributions, as shown in equation 8.8. These results were compared with those for the full CQL algorithm, which is equivalent to the T-CQL algorithm with all thresholds equal to zero. Several threshold distributions were considered, with the parameter a, in equation 8.8, taking values of 0(flat), 2, 4, and 6. The performance is shown in Figure 8.19, see Table 8.2 for details of variances.

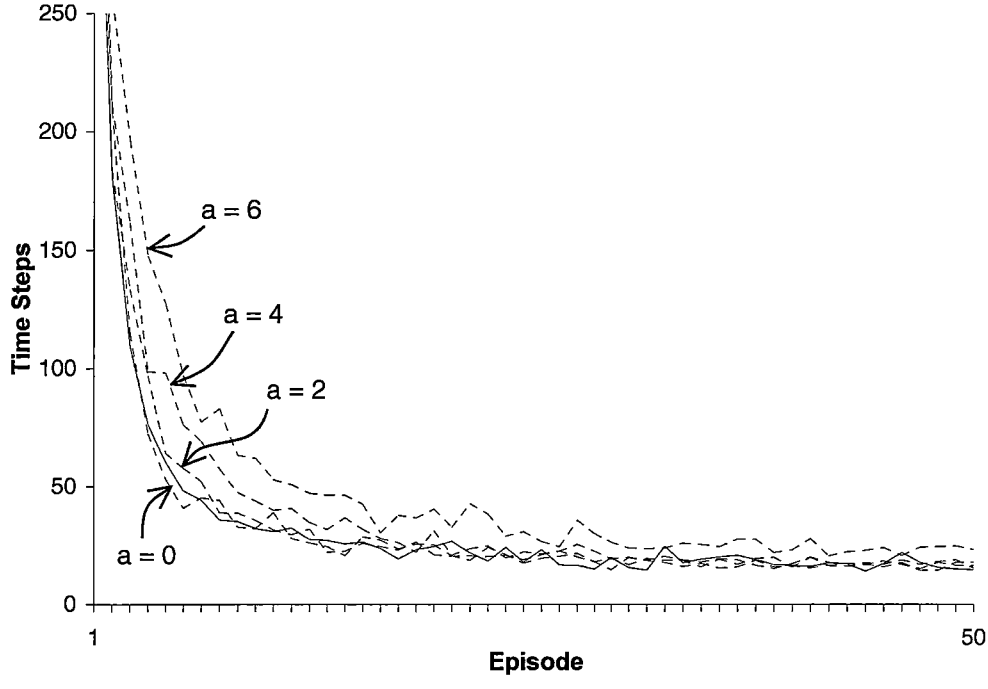

Figure 8.19: Performance comparison of T-CQL and CQL. The solid line shows the performance of the full CQL algorithm with dotted lines showing the performance of T-CQL with thresholds chosen randomly from exponential distributions with the parameter a as shown (error bars have been omitted for clarity).

Table 8.2 Mean time and 95% confidence intervals for CQL and T-CQL.

	Trial 1	Trial 10	Trail 20	Trail 30	Trail 40	Trail 50
Complete	380±32	32±9	25±7	20±5	17±2	15±2
$a = 0$	380±26	32±7	25±6	15±1	16±2	16±2
$a = 2$	403±27	36±8	21±4	20±5	16±2	18±3
$a = 4$	435±32	44±9	31±7	19±5	17±3	15±2
$a = 6$	450±31	62±11	41±9	26±5	23±4	23±5

Figure 8.19 shows that the final performance, of all but the most extreme (a=6) threshold selections, was comparable to the full version of CQL. The learning rate for a=0 and a=2 appear comparable to the full version, with some indication of a slight decline as a increases further.

Figure 8.20 shows the average number of updates required per time step for the same set of threshold distributions. T-CQL performed approximately 1/8th, 1/25th, 1/60th and 1/150th the number of updates compared to CQL for values of a equal to 0, 2, 4 and 6 respectively. Of all threshold distributions tested, only a=6 came close to reaching its theoretical maximum number of updates in this environment.

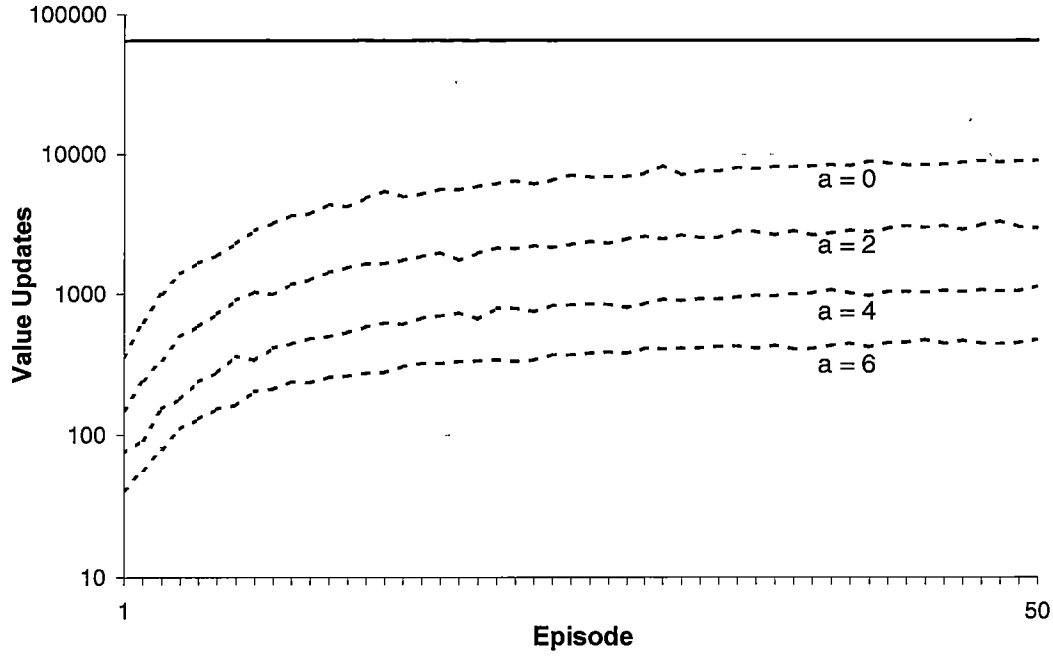

Figure 8.20: The average number of updates made per time step for each episode. The solid line is for the full version of CQL; dotted lines show the values for T-CQL for the given threshold distributions. (Error bars for these graphs are insignificantly small and have been omitted)

In order to get an indication of how well the performance of T-CQL scales as the number of states increases, the observed path length was also compared to the optimal path length for each threshold distribution. The results are shown in Figure 8.21, and demonstrate that, for conservative threshold distributions, T-CQL should scale well as the number of states, and hence average path length, increases. However, for an exponential threshold distribution with a=6, the performance degraded rapidly as the goal distance increased. However, given the R^2-values for trend-lines in Figure 8.21, it is difficult to draw any strong conclusions from this limited data.

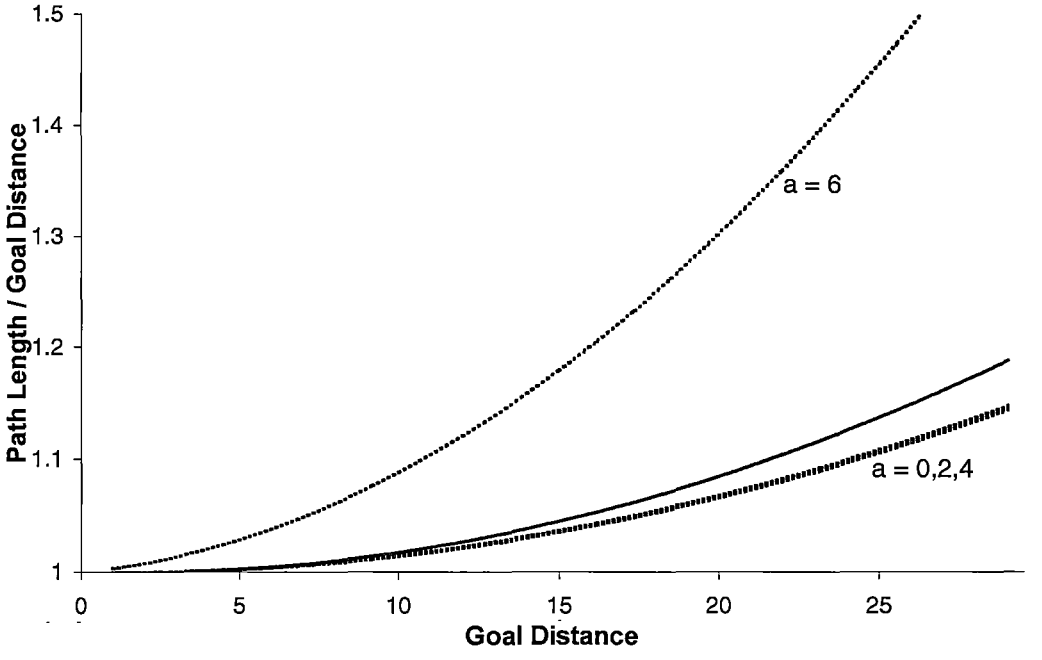

Figure 8.21: T-CQL scaling as goal distance increases. The solid line shows the trend line for the complete version of CQL; dotted lines show trends for T-CQL. Trend-lines for a equal to 0, 2 and 4 are indistinguishable. The R^2-values for these trend-lines are 0.629, 0.685, 0.760, 0.712 and 0.531, for complete CQL and T-CQL with a equal to 0, 2, 4, and 6 respectively.

8.3. Summary

In this chapter, a new algorithm called concurrent Q-learning was developed to enable robust navigation in complex and dynamic environments with dynamic goals. The new algorithm is similar to DG-learning, but action values are fully utilised to enable more value updates for each experience. Unlike the relaxation procedure used in DG-learning, these updates are able to deal effectively with novel obstructions or optimistic value initialisation.

The main weakness, of both DG-learning and concurrent Q-learning, is the poor update time complexity. A hierarchical form of CQL, called T-CQL, was developed to address this issue. T-CQL is able to perform updates with time complexity $O(|A|)$, with minimal loss in performance. While hierarchical forms of DG-learning do exist, these still suffer from DG's inability to deal with novel obstructions.

Chapter 9. System Integration and Testing

In previous chapters the localisation and navigational systems were developed. Chapter 5 developed view cells which are used as input for the place cell system developed in Chapter 6. Chapter 7 developed the low-level navigational system responsible for collision avoidance and simple exploratory behaviour, and Chapter 8 developed the high-level system responsible for path planning. This chapter discusses the integration of the sub-systems (Section 9.1), initialisation and pre-training (Section 9.2), and finally presents the performance of the complete system (Section 9.3).

9.1. Integration

Some components of the final system have already been tested together in previous chapters. The view cell system of Chapter 5 was used as input for the low-level, collision avoidance system developed in section Chapter 7. The view cell system also provides crucial input to the place cell system developed in Chapter 6. Figure 9.1 is a diagram of the complete system showing these interactions, and also showing other components that need to be integrated.

Figure 9.1 shows that, in addition to the interactions already discussed, the high-level navigation system receives the current location as input from the place cell system. Therefore, the current goal location must also be in the same place cell format, and this conversion is made by the goal system, which also receives place cell input for training. Output from the path planning system is in the form of the preferred heading, and this is converted to a preferred motor action. Finally, conflicts between the preferred action from the path planning system, and the safe exploratory action from the low-level navigation system, are resolved by the motor control unit.

This section discusses the goal system, the conversion of continuous place cell input into a discrete format suitable for use by the CQL algorithm, and the implementation of the motor control unit.

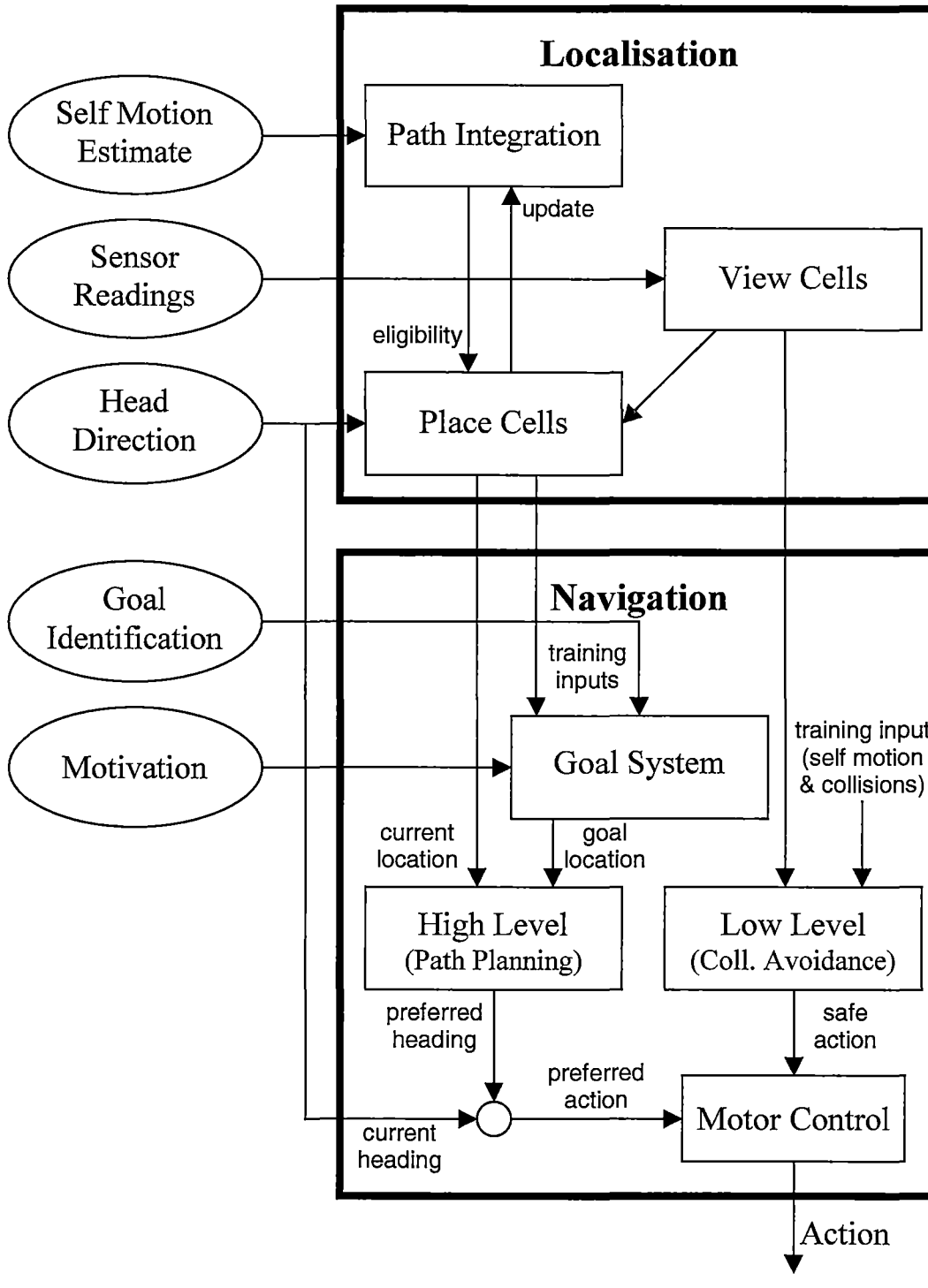

Figure 9.1: Schematic diagram of the proposed model. Ovals represent systems external to the model; boxes represent internal systems.

9.1.1. Goal Memory

A goal memory is needed to learn associations between goals and locations. If the goal is to recharge, for example, the robot needs to learn the locations of all recharging points. Typically, the region where a goal is present will correspond to

one or more place fields, and these fields may or may not be adjacent or overlapping. That is, in general there is no one-to-one mapping from goals to place cells.

A suitable goal system can be implemented using a simple associative memory (Figure 9.2) that learns a mapping between goals and place cells. The memory consists of two layers, a layer of goal cells and a layer corresponding to the place cells, with complete connectivity between the layers. When a goal is encountered, the connection weights between the corresponding goal cell and active place cells are increased, and connections to inactive place cells are decreased (see equation 9.1). To facilitate latent learning, this training is performed even when the goal encountered is not the current goal.

$$\Delta w_{ij} = \eta_G \, GC_i \left(PC_j - 0.5 \right) \qquad \textbf{9.1}$$

When choosing an action, the input of this memory is set to the expected reward for achieving each goal (typically, the reward for one goal will be 1, while the reward for other goals will be 0). The output of the memory is then taken to be the expected reward, r^{s^d}, for reaching each place cell or state s^d. Actions can then be selected using the action selection algorithm in Figure 8.5.

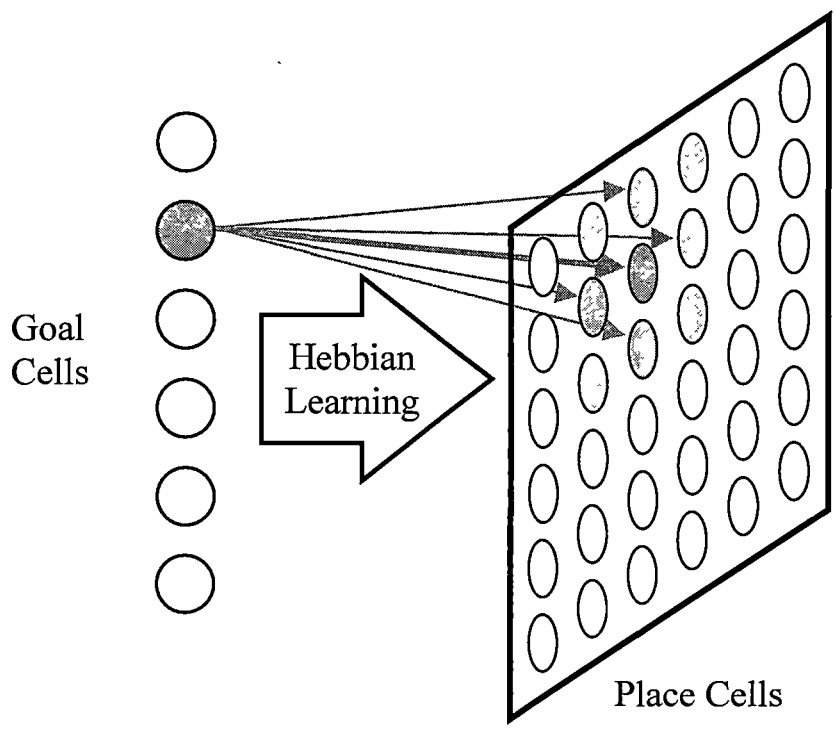

Figure 9.2: Goal-place associative memory.

9.1.2. Planning Updates

The goal system provides the motivational input to the planning system and this allows actions to be chosen. However, a suitable method for updating the planning system is required. To do this we must choose when and how to perform temporal difference updates.

If the winning place cell were taken as the current location, then the system could be updated when a new winning cell was observed. However, due to the continuous nature of place cell activity, and noisy input from the view cells, the winning place cell may change rapidly even when the robot is not moving. This will lead to many unnecessary and ultimately pointless, if not detrimental, updates. To solve this problem a form of hysteresis was introduced. Updates were performed only when the output of the winning place cell was significantly (50%) higher than the current output of the cell that was the winner at the time of the previous update.

As well as the current state, the path planning system needs to know the previously chosen action. Due to the nature of the environment and possible conflict with the collision avoidance system, the previously selected action may not be related to the action ultimately performed. For example, the planning system may suggest movement in a particular direction, but the collision avoidance system may predict a collision for movement in that direction. In this case, if the system decides to do nothing, the planning system would never be updated, since the winning place cell would never change. On the other hand, if an alternate action were chosen (see Figure 9.3), the system would have to decide which action should be used to update the planning system.

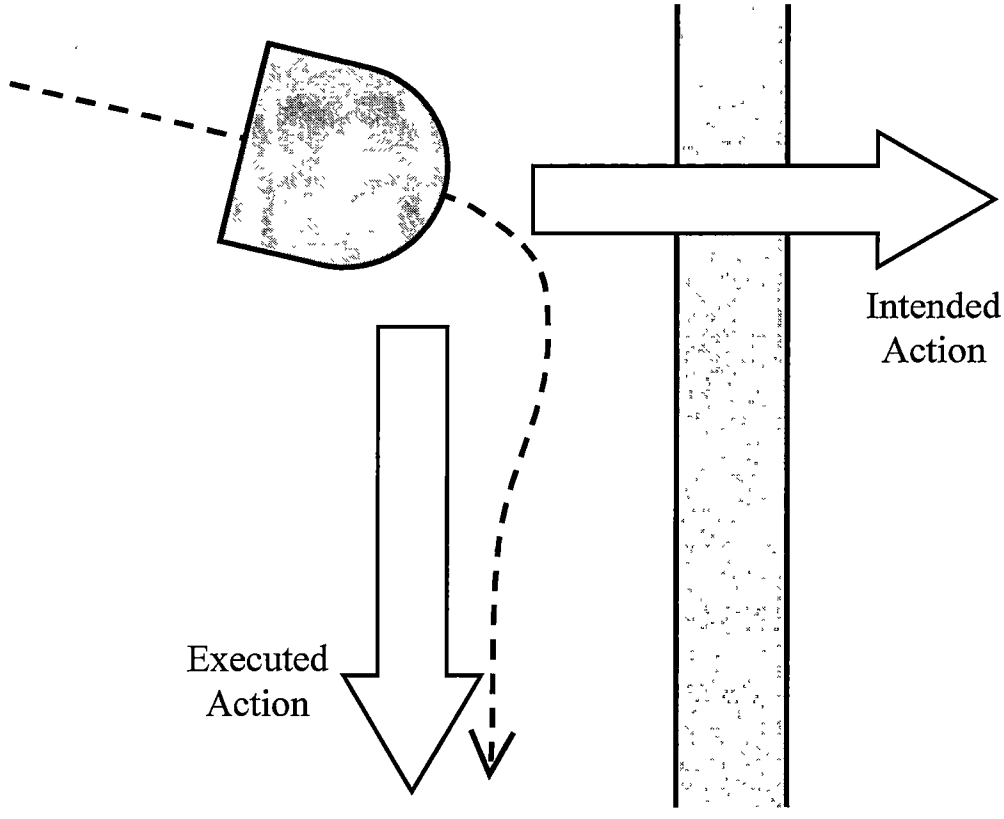

Figure 9.3: A situation where the action chosen by the path planning system may not be the best action to use for updating the system. The dotted line shows the actual path followed by the robot.

If the actual direction of movement were used for training, the planning system would never learn anything about the intended direction of movement and may continue to choose that action in future. Alternatively, if the intended action were used for training then the planning system would learn an erroneous and possibly unpredictable outcome for that action. Finding a robust solution to this apparently simple problem proved to be a non-trivial task.

It was decided that a combination of these alternatives should be used. If the motor system decides that remaining stationary is the best way to resolve conflicting input from the planning and collision avoidance sub-systems, the planning system is trained using the intended action for the update. This update will have the effect of reducing the expected value of that action, since the state has not changed. Alternatively, if another action is performed, as in Figure 9.3, the planning system is updated using the executed action. The system was also initialised with low values so that these alternative actions will quickly become more attractive than those obstructed actions that are never executed, and hence rarely updated.

9.1.3. Combining Planning and Low-Level Navigational Input

The motor system receives input from the planning system, the collision avoidance system, and the head direction system. The desired direction of movement from the planning system is compared to the current heading from the head direction system. This relative direction is then used to modify the action values provided by the collision avoidance system. Figure 9.4 shows the action values to be modified.

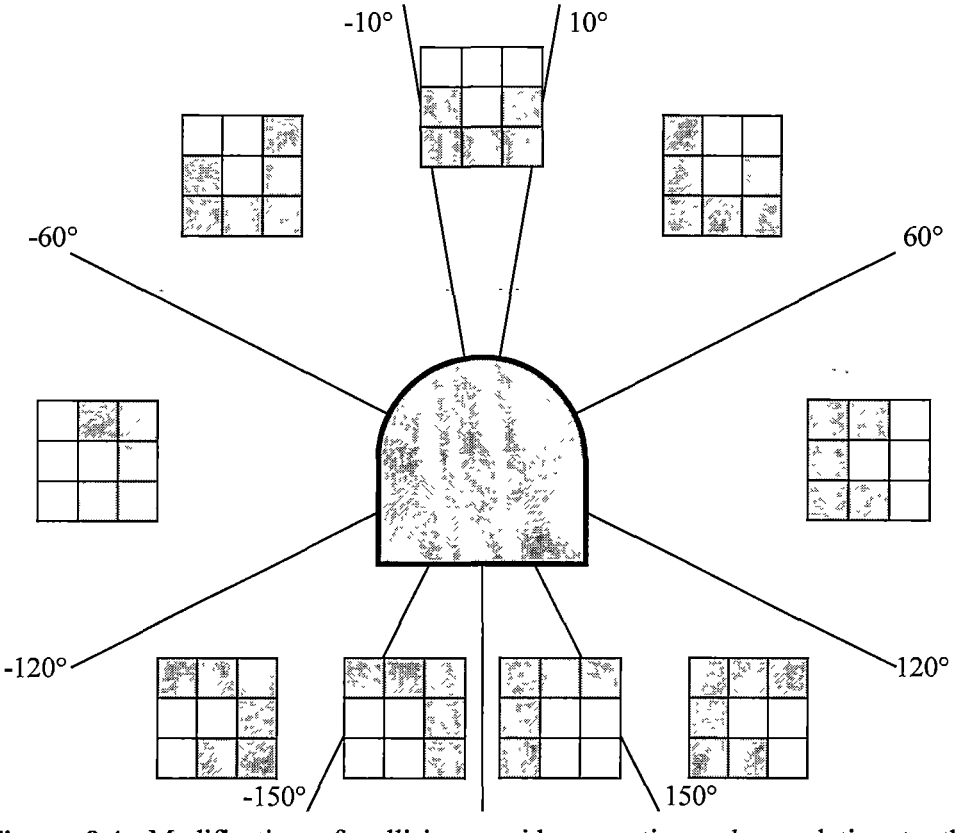

Figure 9.4: Modification of collision avoidance action values relative to the preferred direction of movement suggested by the planning system. The 3×3 grids represent the nine action values for moving (from left-to-right and top-to-bottom) forward and turning left, forward, forward and turning right, turning left, stopping, turning right, backwards and turning left, backwards, and backwards and turning right. Each grid shows the values for movement if the preferred direction is in that sector relative to the heading of the robot. The shaded boxes represent action values to be reduced.

If the planning system does not know how to reach the current goal, the agent should generally follow the action values specified by the collision avoidance sub-system, since this will achieve rapid exploration of the environment. This should also be true if the goal is not believed to be accessible or if no goal is specified. On the other hand, if the goal is known to be accessible, the action values specified in Figure 9.4 should be significantly reduced to encourage movement in the appropriate direction.

However even in this case, the collision avoidance system should be able to suggest small trajectory excursions to manoeuvre around obstacles or to account for small errors in the position estimate.

This behaviour is achieved by reducing the indicated values by an amount depending on the planning system's predicted return for reaching the current goal, g, as shown in equation 9.2.

$$\text{CA modifier} = 1 - \frac{R^g(s,a)}{5} \qquad \textbf{9.2}$$

When the goal location is near, the route chosen by the planning system will be closely adhered to, whereas for a more distant goal, exploratory actions are more likely to occur. If the planning system is initialised with low values based on the relative positions of place fields, these will provide a small preference for moving in the chosen direction. However if a wall is encountered, this modification will not be large enough to stall the exploration process.

9.2. Pre-training and Initialisation

A large part of the navigational system may be pre-trained. View cells should generalise well enough to enable the same set of cells to be used across many environments. Similarly, the collision avoidance system may also be pre-trained, since input to this system is view cell output. In addition to this pre-training, one important advantage of using a place cell system with fixed place fields is that this preconfiguration should allow for intelligent initialisation of the path planning system.

9.2.1. View Cells and Low-level Navigation

To test the usefulness of pre-training, the view cell and collision avoidance sub-systems were trained in the environment shown in Figure 9.5a. The robot was then transferred to a new environment, shown in Figure 9.5b, without re-initialising the view cell and collision avoidance systems. The average reward received by the low-level navigation system in the second environment was compared to the average in the first environment after training was complete.

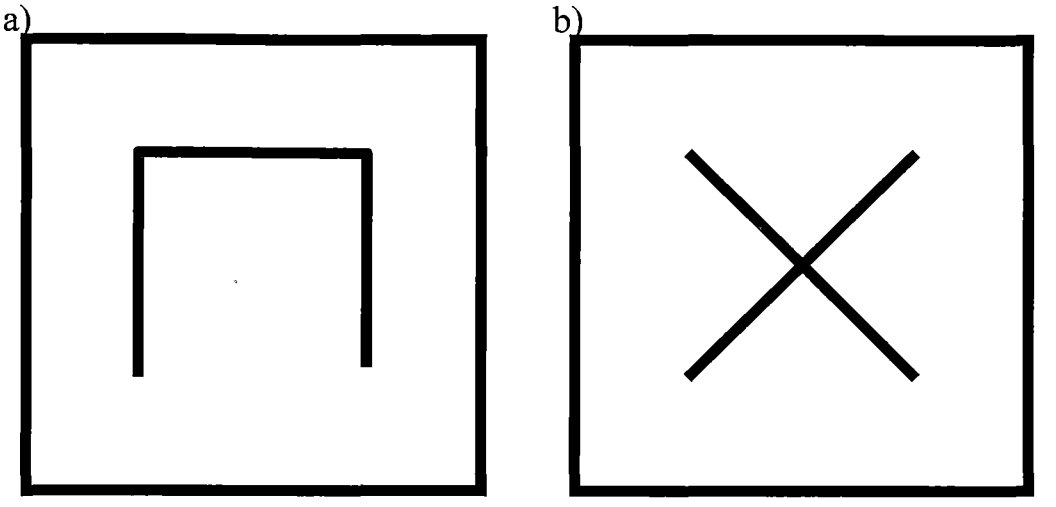

Figure 9.5: Environments used for testing the view cell and collision avoidance pre-training. a) is the training environment and b) is the test environment.

In the training environment the average reward per time step was 0.831±0.005 (95% confidence), compared with 0.833±0.005 (95% confidence) in the test environment. A more dramatic change to the environment might yield poorer results. However, provided a suitably varied training environment is chosen, these results indicate that pre-training is unlikely to have a significant adverse effect on performance. In fact, in the majority of situations, pre-training should significantly improve learning speed in new environments. Therefore, all remaining experiments were conducted after these systems were trained in the environment depicted in Figure 9.5a.

9.2.2. Place Cells and High-Level Navigation

The view cell to place cell connections cannot be pre-trained and, since no information about the presence of obstructions is known prior to entering the environment, the path planning system cannot be pre-trained. However, since place field centres are fixed for each place cell, the path planning system can be initialised using this information. An initial estimate of action values can be made for each pair of place cells based on the distance between their place fields (see Figure 9.6). For an open environment with no obstructions, this initialisation alone would solve the path planning problem. However in more complex environments, this initialisation may be detrimental, especially if dead-ends are present.

It should be noted however, that for an open environment the action values themselves are not important. Instead, it is the value of each action relative to other action values that is important. Therefore, the action values may be initialised with low, but spatially consistent, values. That is, the action values of a pair of place cells

may be initialised using some multiple of the distance between the place field centres as shown in equation 9.3.

$$Q^{s_2}(s_1, a) = \gamma^{\left(\frac{m\left[\bar{p}_2 - (\bar{p}_1 + \bar{x}_a)\right]}{\|\bar{x}_a\|}\right)} \qquad \textbf{9.3}$$

where m is the scaling factor, $\bar{p}_1$ and $\bar{p}_2$ are the place field centres for place cells corresponding to states s_1 and s_2 respectively, and $\bar{x}_a$ is the expected displacement vector for action a (and will have the same magnitude for all actions), as shown in Figure 9.6.

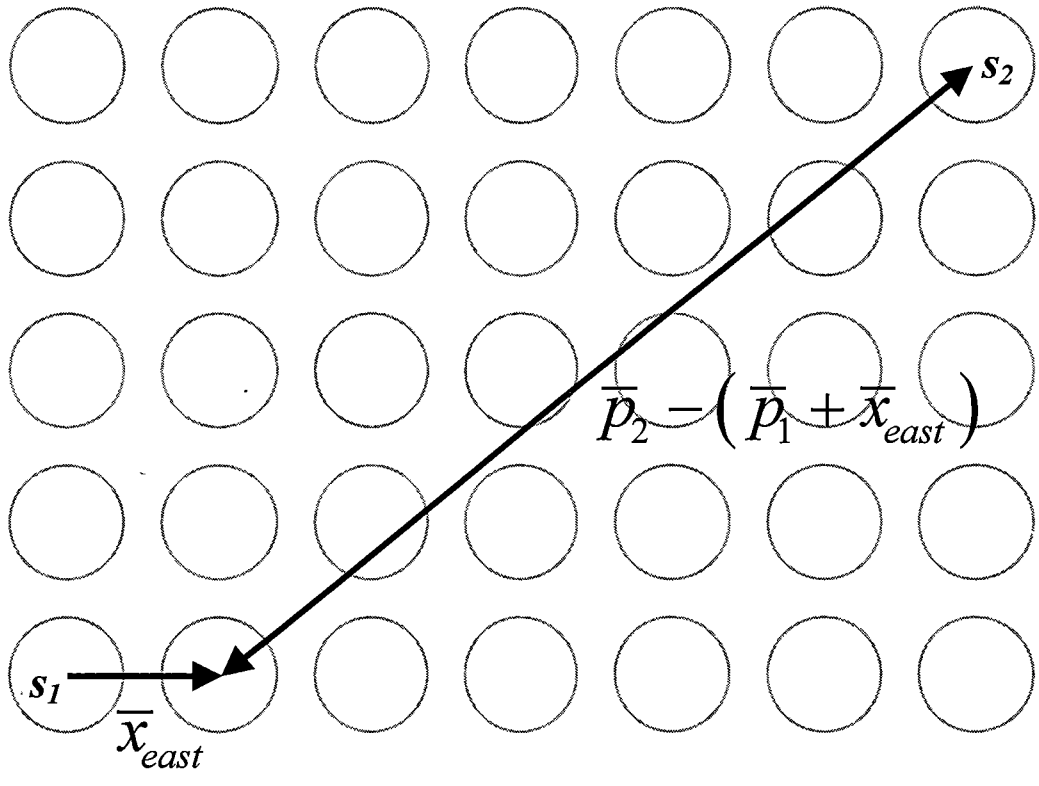

Figure 9.6: Example demonstrating the procedure for the calculation of action value initialisation.

Such an initialisation will enable immediate navigation in open environments, while allowing any errors in complex environments to be quickly corrected. In addition, once a valid path is found, the corresponding action values will dominate this initialisation.

This initialisation has the added benefit of greatly increasing the number of action values that may be updated in the early stages of training, as illustrated in Figure 9.7. In Figure 9.7a, only the action just performed is updated, whereas in Figure 9.7b, all

paths containing this state transition are updated. Note that such an initialisation would not be suitable for DG-learning, since the relaxation process will not be able to decrease these values if an obstruction is encountered. Whereas, CQL will make the appropriate updates for negative as well as positive errors.

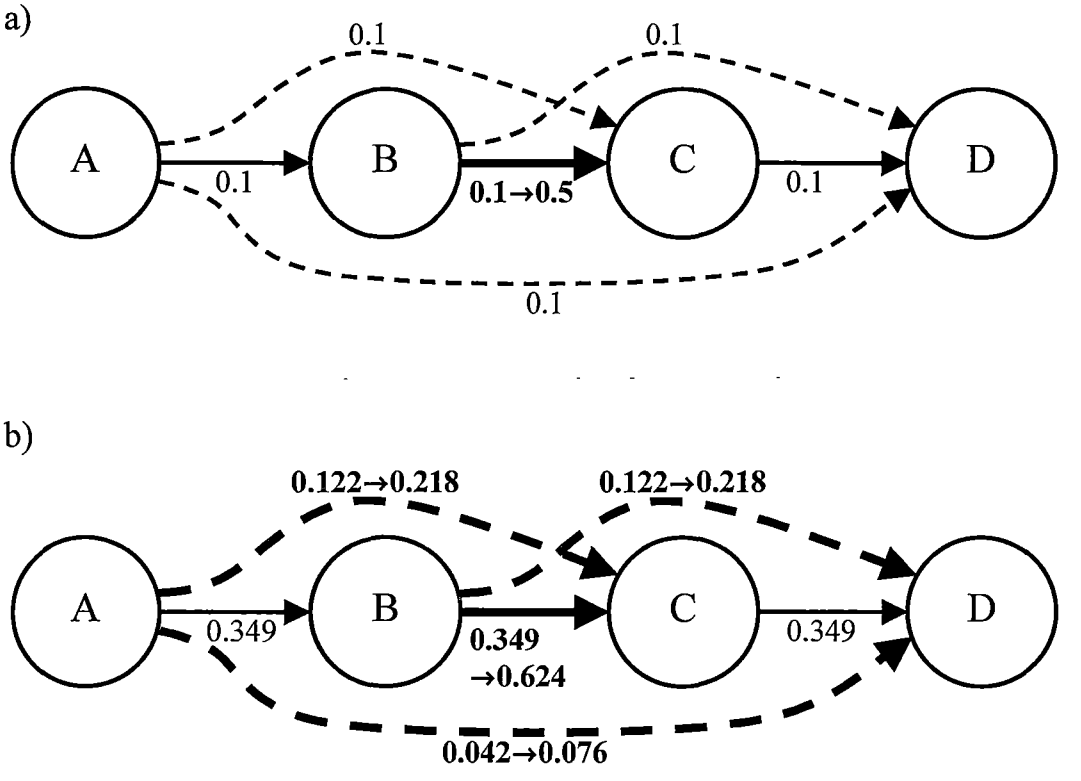

Figure 9.7: An environment includes four states A, B, C, and D, with direct movement possible (solid arrows) from A-B, B-C and C-D. Dashed arrows indicate action values for non-direct movement. An agent starts at B and moves to C. a) Shows the possible value updates, for $\gamma = 0.9$ and $\eta = 0.5$, if values are uniformly initialised with the value 0.1. b) Shows the possible updates if values are initialised with low, but consistent, values. In this case, the value of m in equation 9.3 is 10 (ie. $0.349 = \gamma^{1\times10}$, $0.122 = \gamma^{2\times10}$, etc.).

The degree to which interstate distances should be overestimated for the purposes of initialisation is likely to depend on the environment. If the environment is known to be completely open, for example, the navigation system should be initialised using the true distances, and minimal further training will be required. As the complexity of the environment increases, it is likely that the distances should be increasingly overestimated by increasing the value of m in equation 9.3.

This hypothesis was tested by training the robot in four environments of varying complexity, shown in Figure 9.8. The robot was asked to find each of the eight goal

locations, followed by a further 42 goals randomly selected from these eight. The robot was not relocated after successfully finding a goal.

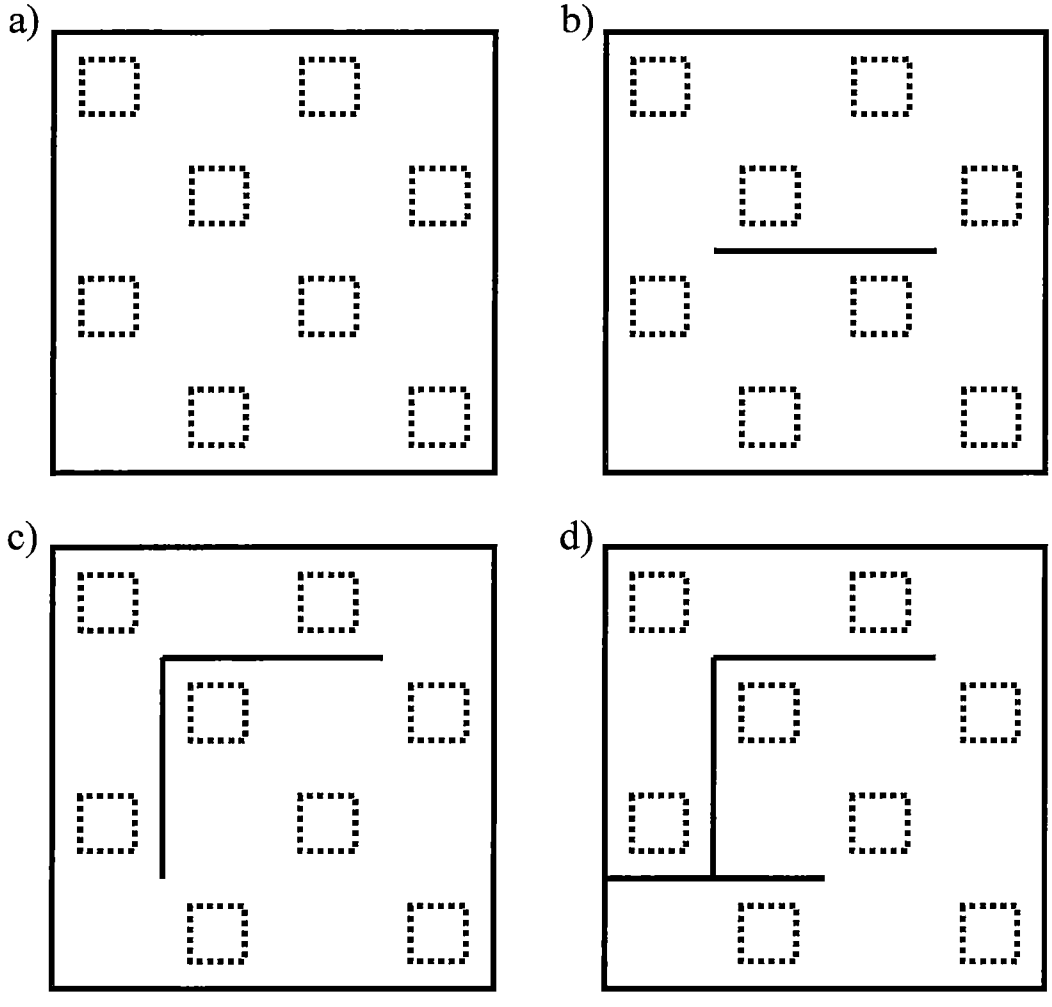

Figure 9.8: Environments used for testing initialisation of the path planning system. Solid lines represent walls; dotted squares represent goal locations.

Figure 9.9 shows the performance of the robot for different value initialisations. As expected, the best value for the initialisation parameter, *m*, increases as the complexity of the environment increases. For the open environment (a), the best value is approximately 2 to 4. Surprisingly, using a scaling factor of 1 (the actual interstate distance) for initialisation does not give optimal results. This is presumably due to the fact that movement and position estimates are not precise, since the robot may not be at the exact centre of the place field when updates are made. For environments b, c, and d, the best scaling factors are 4, 16, and 32 respectively. A scaling factor of 16 to 32 gives reasonable performance across all environments.

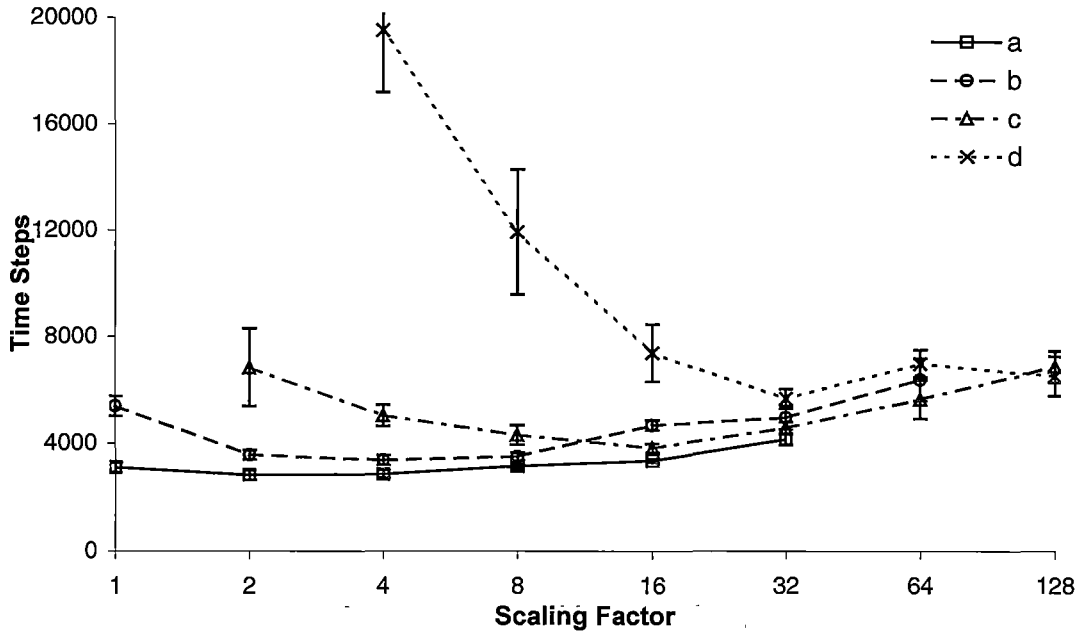

Figure 9.9: Performance in each of the environments a, b, c and d, showing the number of times steps taken to find 50 goals for various values of the scaling factor *m*.

9.3. The Complete System

The quantitative results in the previous section do not give a good indication of the actual paths chosen by the robot. This section presents actual paths chosen by the robot for the complete system, using both pre-training and path-planning initialisation. In particular, environments and tasks were chosen to help characterise the nature of the robot's navigational decision making. This is very important for any robot that is expected to interact with humans, as is often the case for mobile robots. In such cases, a robot that makes a predictable error may be more desirable than a robot that makes a technically correct but unexpected decision.

Figure 9.10 shows the effect of the initialisation on navigation in an open environment. The robot starts at a goal location and proceeds to explore the environment. At some point, the robot is asked to return to the starting goal location. In such a task a typical human would anticipate that the robot would return to the initial location via the shortest route, rather than retracing its original path.

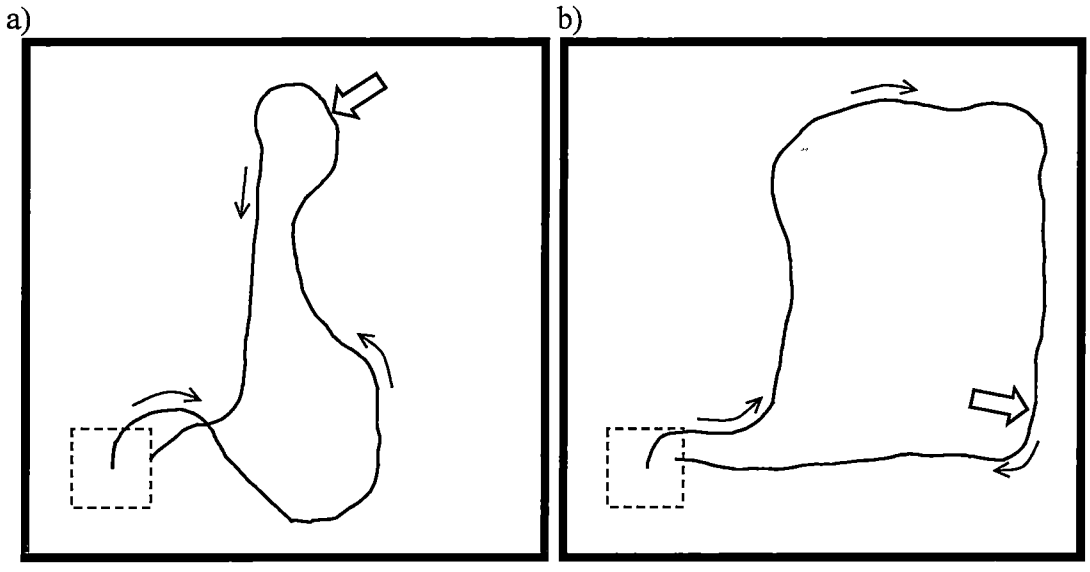

Figure 9.10: Analysis of system dead-reckoning ability. The robot starts at a goal location (dashed square) in the southwest corner of the environment. At some point (indicated by the block arrow) the robot is instructed to return to the start location.

The paths in Figure 9.10 demonstrate the value of the initialisation procedure discussed in the previous section. When the robot is told to return to the original location, a direct route is chosen. With standard initialisation, such as would be required if place field centres were not fixed, the robot would either return via the original route, or randomly attempt to find some other route. Neither of these options would inspire confidence in a human observer or operator.

Figure 9.11 and Figure 9.12 demonstrate learning in a more complex environment where pre-initialisation may lead to poor performance. The task and environment are identical to those used in Figure 9.8d. The robot is asked to move to a randomly chosen goal location. Upon reaching this goal, a new objective is randomly selected and the robot continues to the next goal. For this experiment, 100 consecutive goals were used.

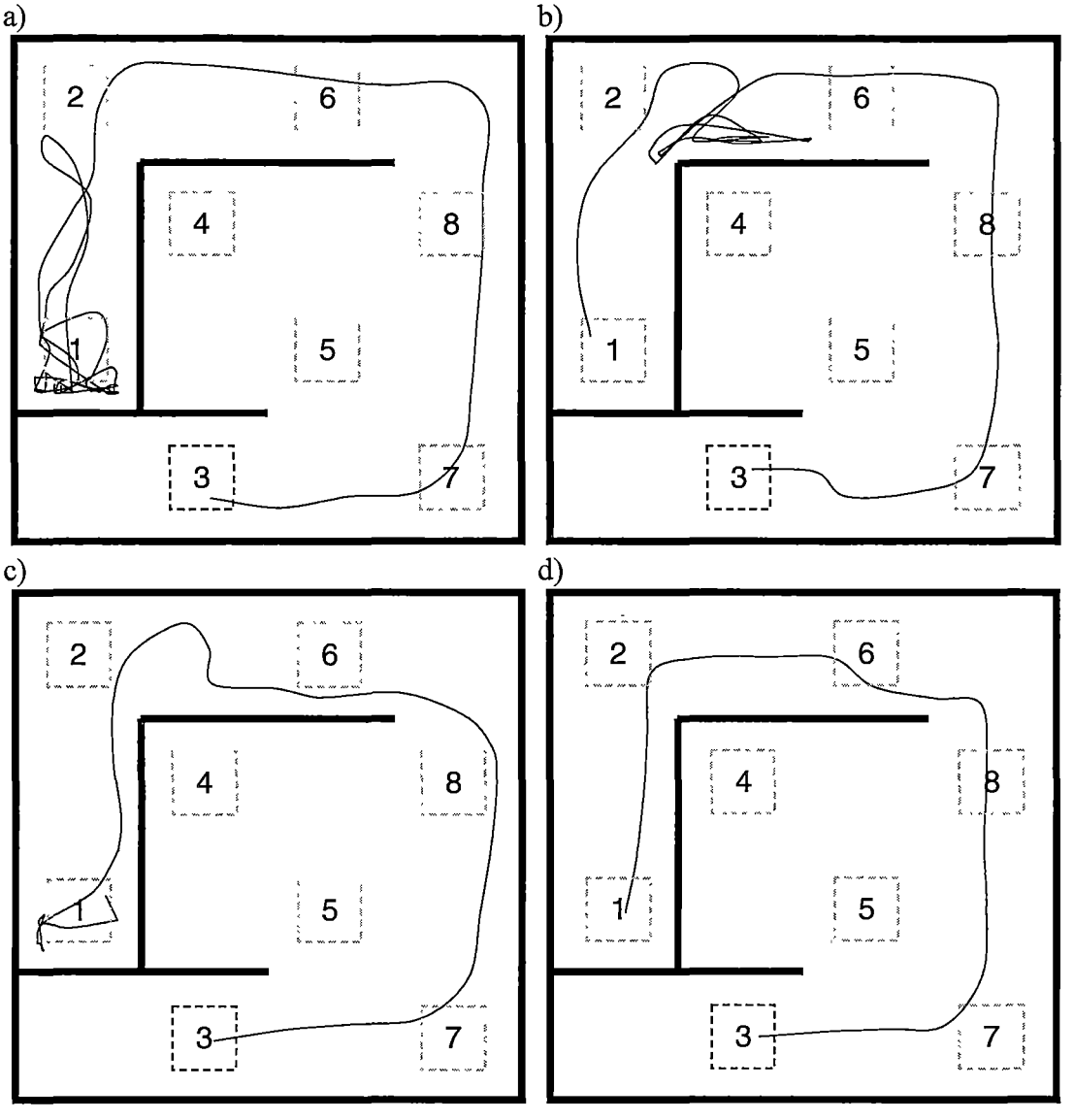

Figure 9.11: Learning performance of the complete navigational system, showing the path chosen by the robot when navigating from goal 1 to goal 3 at various stages of training. a) shows the first attempt, with successive trials shown in b) through d).

In Figure 9.11a, the robot has just moved from the starting location in the southwest corner to goal location 1 via a circuitous route that passed through goal 3, hence learning the location of goal 3 en route. The robot is then asked to return to goal 3. Despite having learned one possible route to goal 3, the robot attempts the shorter, direct path indicated by initial action values. A number of possible actions in this area have favourable values for reaching locations near goal 3, and several of these are tried before resorting to the longer path already discovered. Figure 9.11b shows the path chosen by the robot on the next occasion that the robot reached goal 1, and was asked to navigate to goal 3. On this occasion the robot immediately begins taking the known longer route, but then attempts to find a shorter path near goals 2

and 4, a region not explored completely on previous trials. After a short time, the robot continues on the known route. On the third occasion that the robot is faced with the same problem (Figure 9.11c), the robot decides to use the known route after briefly checking for shortcuts near goal 1. Finally, Figure 9.11d shows that on the fourth occasion the known route is chosen immediately.

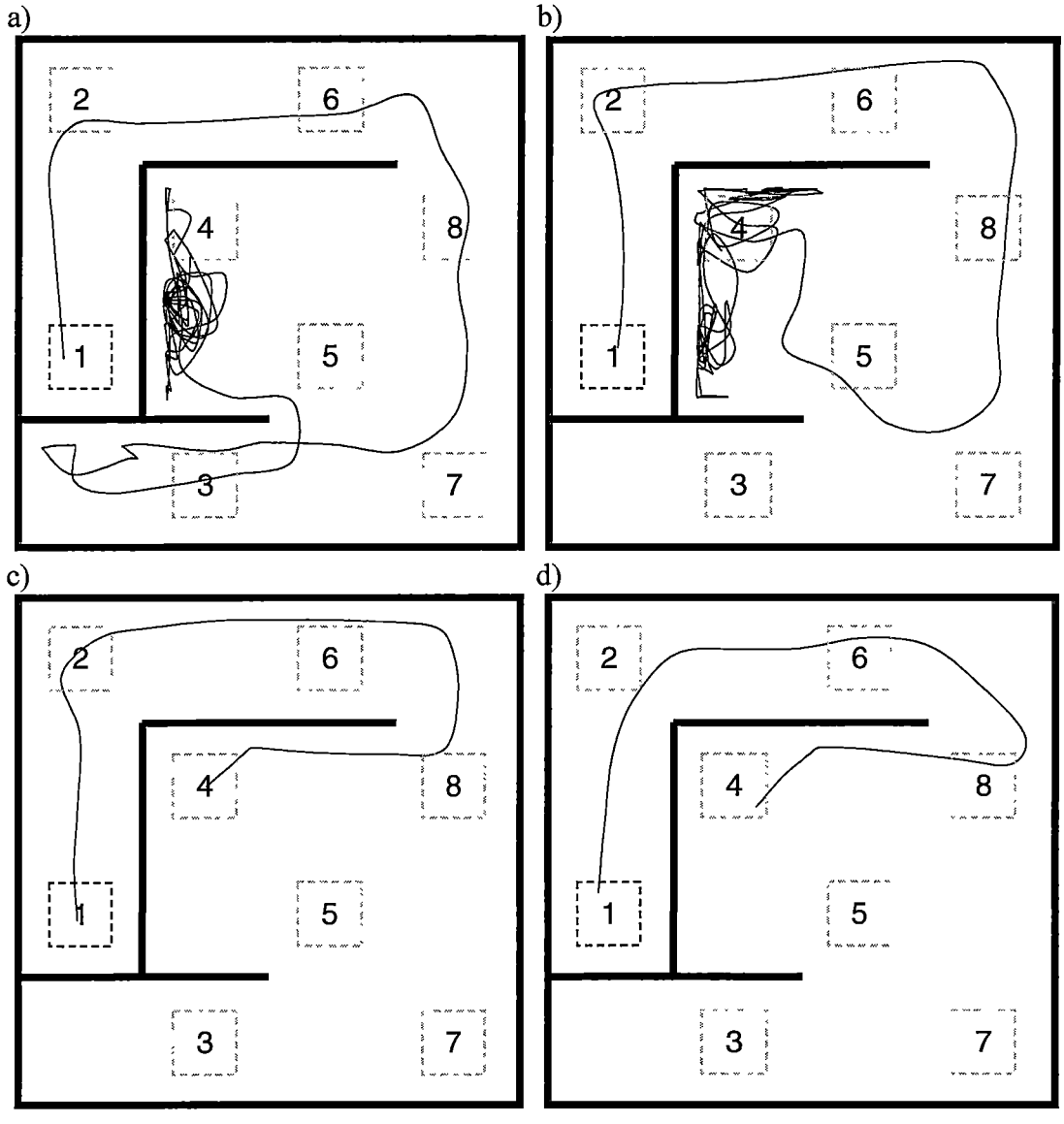

Figure 9.12: Learning performance of the complete navigational system, showing the path chosen by the robot when navigating from goal 4 to goal 1 at various stages of training. a) shows the first attempt, with successive trials shown in b) through d).

Figure 9.12 shows results from the same experiment for paths from goal 4 to goal 1. In Figure 9.12a it can be seen that, under some circumstances, the robot may explore

widely before reverting to the known route. As possible shortcuts are explored and discounted (Figure 9.12b), the routes become more direct.

While the results of these experiments show that the behaviour is not always optimal, a human observer would get the sense that the robot knew what is was trying to achieve. It would be easy to imagine an animal behaving similarly in a similar situation.

Figure 9.13 demonstrates the behaviour of the robot in dynamic environments. The robot is initially trained in an environment with four goal locations. The environment consists of two rooms separated by two doors, similar to the environment in section 8.2.3. At some point, one of the doors is closed and the robot must learn to use the alternate route. This task is similar to the detour experiment in section 8.2.4. Here the robot is tested in a continuous environment, and with the benefit of pre-training and initialisation.

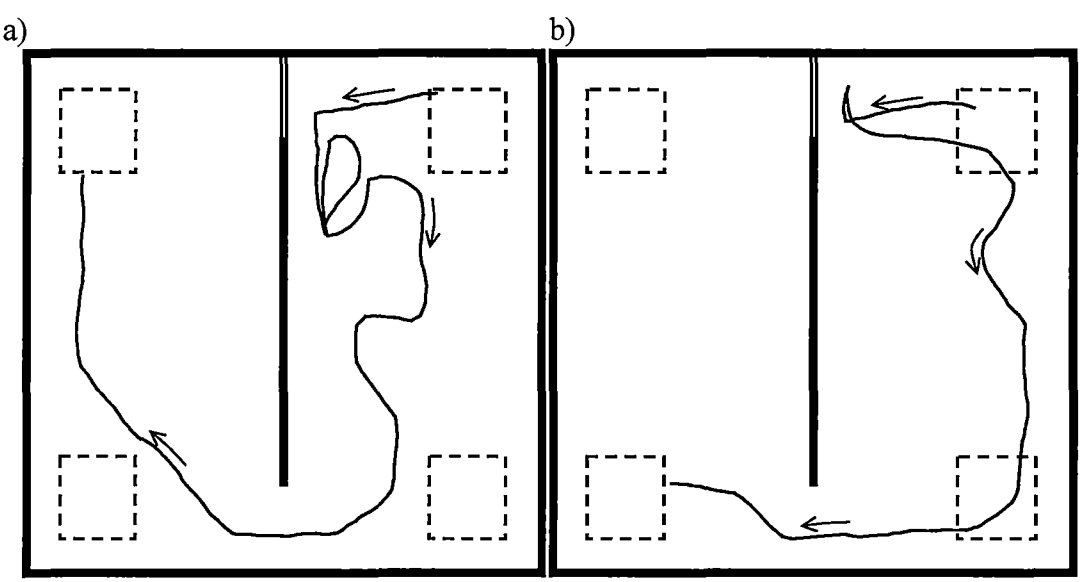

Figure 9.13: Performance in dynamic environments. Dashed squares indicate goal locations. Thick solid lines indicate walls. The double lines in the north of the environment indicate a closed door.

In Figure 9.13a, the north door is closed once the robot reaches the northeast goal location. The robot is then asked to navigate to the northwest goal. The robot initially attempts to use the north door. As action values for reaching the northwest goal via the north door drop, the robot moves a short distance away to try to find a route through the nearby wall, which may not have been attempted earlier. The robot then returns to the door and tries that route again. As these action values continue to drop, the robot finds that the route via the south door is more favourable.

In Figure 9.13b, the robot is asked to navigate to the southwest goal when the door is closed. The robot chooses the route via the north door, although the southern route is equal in length. Since both routes are of similar length, the robot changes to the southern route more quickly than in Figure 9.13a. Note that the route chosen is via the more familiar path from the northeast goal to the southeast goal.

It is difficult to assess the performance of a real or simulated robot, since simple measures of path length or travel time do not capture the subjective qualities of the robot's performance. These subjective qualities can have a significant impact on how the robot is perceived by non-academic or casual observers, and in many cases this is of significant importance. These experiments show that the paths chosen by the robot appear intelligent and reasoned. While the paths may not be optimal, it is easy to rationalise about why a particular path is chosen and this is a very important quality for any robot that is intended to interact with humans.

Chapter 10. Conclusion

This thesis has presented a localisation and navigational system for a simulated mobile robot based closely on biological theories and experimental results. The system was shown to be capable of efficient and robust navigation in complex and dynamic environments. While the system is yet to be verified in a real environment, this work has already resulted in new learning algorithms for localisation and navigation, that may also be applied to other problems in the field of artificial intelligence.

10.1. Localisation

Localisation was achieved first through the establishment of view cells, cells that respond to a particular sensory view. In the case of these experiments, the sensory view consisted of the readings from an array of range sensors. The sensory pattern from such an array is highly dependent on the position and especially the orientation of the robot. For example, depending on the angle of incidence with a wall, the range returned by a sensor may change either dramatically with the robot's orientation, or very little. For every given view of the environment small changes in position and orientation may result in a significantly different sensory pattern, with each individual sensor affected to differing degrees. This presented a problem for the localisation system since it is desirable for the perceived view to change little for small changes in position and orientation.

To overcome this problem a new type of locally tuned neural model was developed. Adaptive response function neurons provide an online method for learning the centres, widths and shapes of basis functions for locally tuned neurons. These neurons provide a high level of generalisation without loss of class separability. Adaptive response function neurons (ARFNs) may be trained in either a supervised or an unsupervised manner, and should prove to be a useful tool for solving many problems in the field of pattern recognition. This was demonstrated through testing on several standard classification problems where a network of ARFNs was able to achieve similar or better performance than a similar network of locally tuned neurons that learned the centre of the response functions only.

ARFN view cells alone display very useful localisation properties, but suffer from perceptual aliasing. To overcome this problem, a place cell layer was implemented that received input from both the view cells and a simple path integrator. The

preferred location of each place cell was pre-initialised allowing the navigational system to perform dead reckoning without the need for coordinate learning.

10.2. Navigation

Two types of navigational systems were implemented. The low-level navigation system uses standard Q-learning to learn collision avoidance behaviour. View cell input was used as input to this system and was shown to be a better source of input than the raw sensor data. View cell input is particularly useful as the output or the ARFNs is designed to vary little with small changes in position and orientation, thus reducing the possibility of repetitive, contradictory action selections.

The high-level navigation system is used for path finding and this is achieved using the new Concurrent Q-Learning (CLQ) algorithm. CQL is a general reinforcement learning algorithm that is goal independent. Information gained in one task is automatically transferred to new tasks without suffering interference or loss of information pertaining to the original task. Furthermore, CQL makes optimal use of information gained from environmental experiences by updating all possible states after each experience. This improved performance is especially noticeable in dynamic environments. CQL is able to quickly choose detours and find shortcuts as the environment changes. A hierarchical version of CQL was also developed that reduces the time complexity of the update algorithm to such an extent that practical application of the algorithm is not limited by the size of the state space.

The implicit mapping system implemented through CQL will have significant advantages over an explicit map. When learning an explicit map it is imperative that localisation errors are kept to an absolute minimum. Any error in localisation, that is not consistent across the entire environment, will result in an inconsistent map. In contrast, localisation need only be locally consistent with the implicit mapping achieved through reinforcement learning. For example, it does not matter that the perceived heading is North when the actual heading is East; all that matters is that performing a certain action either will or will not take the robot closer to the goal. This has the potential to reduce the need for precise, accurate sensors thus reducing the cost and complexity of any real system.

10.3. Biological Implications

Previous models have addressed many of the problems covered in this thesis. However, while other models have addressed various components of spatial cognition, the current work presents a complete and robust localisation and

navigational system. In addition, the current work presents novel and interesting solutions to some of the problems faced by biological systems. In particular, new insights into dead reckoning and navigational abilities are discussed.

The place cell system presented assumes that place cells have predetermined path integrator coordinates. These coordinates may be used for dead reckoning in new environments with no need for coordinate learning. The current model captures the coordinate information in initial action values that are later modified through reinforcement learning. The agent then has immediate access to this information without the need for vector calculations. While not the case in the current implementation, this information could be distributed across two groups of place cells. One group of place cells would maintain general spatial relationships that are independent of the environment. This group would be closely related to the path integration system. The second group of cells would represent locations in particular environments.

Navigation in the model is achieved using reinforcement learning, which is presumed to be the function of the basal ganglia. While other researchers have postulated a role for the basal ganglia, and hence reinforcement learning, in navigation, previous models do not propose a specific mechanism. According to the current model, both groups of place cells would be expected to send information to this area. This information would then be processed through methods similar to the concurrent reinforcement learning algorithm developed in this thesis.

This suggests that the hippocampal system is predominantly a mechanism for path integration and localisation, and is not necessarily the location of a cognitive map. Instead, the basal ganglia would form a crucial component of the cognitive map, perhaps in conjunction with the hippocampus. The model also predicts that, if connections between the environment-independent set of place cells and the basal ganglia were severed, the animal would not be able to navigate using dead reckoning. Whereas, if the connection between the second group of place cells and the basal ganglia were severed, the animal would exhibit poor navigation in complex environments.

In the developed model, egocentric view cells also play an important role in navigation, providing valuable input for low-level navigation. The low-level navigation is also achieved through reinforcement learning. Therefore, it is predicted that these cells would also send output to the basal ganglia.

While the focus of this thesis has been on navigation, the hippocampus is thought to play an important role in cognition in general. One hypothesis is that the hippocampus forms a working memory that may play a key role in learning relationships that may later be transferred to long term memory. The current work is compatible with this hypothesis, and therefore the above observations are equally applicable to the field of cognitive neuroscience in general.

10.4. Future Work

While the current system presents a complete localisation and navigational system, it is limited to a single environment (with the exception of view cells and low-level navigation, which are environment-independent). Also, only one set of place cells is modelled, and the system does not propose a role for the various groups of place cells found in animals. In other words, the place cell system developed models only those place cells closely associated with the path integrator. A second group of place cells could be added to the model to represent places in different environments or reference frames. This also suggests a possible separation of action values corresponding to each of the place cell groups. Those place cells closely associated with path integration could store the initial action values described in section 9.2. These action values would not require training. A separate set of action values, for the second group of place cells, could be learned for each environment. The agent could then choose which set of action values to use for navigation in a given situation, in a similar way to the system proposed by Foster and colleagues (2000).

The place cell system used by the model relies on input from egocentric view cells. Combined with the fixed initialisation of place field centres, this leads to bimodal place fields in some instances. While this does not appear to be a significant problem, there may be instances where this causes erratic navigational behaviour. In addition, these unused place cells lead to reduced efficiency. A possible improvement would involve the use of allocentric view cells, either replacing, or in addition to, the egocentric view cells. This could be combined with a method for shifting the place field centres as the environment is explored to achieve a more efficient spread of place cells across the environment. This would be particularly suited to the secondary group of place cells discussed above.

The concurrent Q-learning algorithm will also benefit from further investigation. In particular, the thresholding system described in section 8.2.5 requires further work to determine the best method for assigning thresholds. A method for dynamically

assigning thresholds would be particularly useful, and may result in further improvements in the efficiency and performance of the algorithm.

In addition to these algorithmic improvements, the system will also be verified using a real robot with an emphasis on complex and dynamic environments. Testing with different types of environmental sensors (eg. sonar, vision) will also be conducted. Computer games and training simulations are another area where the navigation system, in particular, has great potential. These applications are very demanding and further work will to be conducted to further develop efficient implementations of the CQL algorithm.

References

Albus, J. S. (1971). A theory of cerebellar function. *Mathematical Biosciences, 10*, 25-61.

Alyan, S., & Jander, R. (1994). Short-range homing in the house mouse *Mus musculus*: Stages in the learning of directions. *Animal Behaviour, 48*:2, 285-298.

Amaral, D. G., & Witter, M. P. (1989). The three dimensional organization of the hippocampal formation: A review of anatomical data. *Neuroscience, 31*, 571-591.

Andrews, H. C. (1970). *Computer techniques in image processing*. New York: Academic Press.

Arbib, M. A. (1999). Parietal cortex and hippocampus: from visual affordances to the world graph. In N. Burgess, J. Jeffrey & J. O'Keefe (Eds.), *The hippocampal and parietal foundations of spatial cognition*. Oxford: Oxford University Press.

Arleo, A., & Gerstner, W. (2000). Spatial cognition and neuro-mimetic navigation: A model of hippocampal place cell activity. *Biological Cybernetics, 83*, 287-299.

Arleo, A., & Gerstner, W. (2001). Spatial orientation in navigating agents: Modelling head-direction cells. *Neurocomputing, 38-40*, 1059-1065.

Asada, M., Noda, S., Tawaratsumida, S., & Hosoda, K. (1996). Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine learning, 23*, 279-303.

Balakrishnan, K., Bhatt, R., & Honavar, V. (1998). A computational model of rodent spatial learning and some behavioural experiments. *Proceedings of the 20th Annual Meeting of the Cognitive Science Society*, Mahwah, New Jersey.

Balakrishnan, K., Bousquet, O., & Honavar, V. (1999). Spatial learning and localization in rodents: A computational model of the hippocampus and its implications for mobile robots. *Adaptive Behavior, 7*:2, 173-216.

Balakrishnan, K., & Honavar, V. (1997). Spatial learning for robot localization. *Proceedings of the Second Annual Conference on Genetic Programming.*

Barnes, C. (1988). Spatial learning and memory processes: The search for their neurobiological correlates in the rat. *Trends in Neurosciences, 11*, 163-169.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics, SMC-13*:5, 834-846.

Basset, J. P., & Taube, J. S. (2001). Neural correlates for angular head velocity in the rat dosal tegmental nucleus. *The Journal of Neuroscience, 21*:15, 5740-5751.

Benson, E., Stombaugh, T., Noguchi, N., Will, J., & Reid, J. (1998). An evaluation of a geomagnetic direction sensor for vehicle guidance in precision agriculture applications. An ASAE Meeting Presentation.

Blair, H. T., Lipscomb, B. W., & Sharp, P. E. (1997). Anticipatory time intervals of head-direction cells in the anterior thalamus of the rat: Implications of path integration in the head direction circuit. *The American Physiological Society, 78*:1, 145-159.

Blair, H. T., & Sharp, P. E. (1995). Anticipatory head-direction cells in the anterior thalamus: Evidence for a thalamocortical circuit that integrates angular head motion to compute head direction. *Journal of Neuroscience, 15*, 6260-6270.

Bousquet, O., Balakrishnan, K., & Honavar, V. (1997). Is the hipocampus a Kalman filter? *Proceedings of the Pacific Symposium on Biocomputing.*

Broomhead, D. S., & Lowe, D. (1988). Multivariate functional interpolation and adaptive networks. *Complex Systems, 2*, 321-355.

Brown, M., & Sharp, P. (1995). Simulation of spatial learning in the Morris water maze by a neural network model of the hippocampal formation and nucleus accumbens. *Hippocampus, 5*, 171-188.

Bruzzone, L., & Prieto, D. F. (1999). A Technique for the Selection of Kernel-Function Parameters in RBF Neural Networks for Classification of Remote-Sensing Images. *IEEE Transactions on Geoscience and Remote Sensing, 37*:2, 1179-1184.

Bures, J., Fenton, A., Kaminsky, Y., Rossier, J., Sacchetti, B., & Zinyuk, L. (1999). Dissociation of exteroceptive and idiothetic orientation cues: Effect on hippocampal place cells and place navigation. In N. Burgess, J. Jeffrey & J. O'Keefe (Eds.), *The hippocampal and parietal foundations of spatial cognition*. Oxford: Oxford University Press.

Burgess, N., Donnet, J., & O'Keefe, J. (1996). Robotic and neuronal simulation of hippocampal navigation. *Proceedings of the AISB-97 Workshop on Spatial Reasoning in Mobile Robots and Animals*, University of Manchester.

Burgess, N., Donnet, J., & O'Keefe, J. (1998). Using a mobile robot to test a model of the rat hippocampus. *Connection Science, 10*:3/4, 291-300.

Chomsky, N. (1968). *Language and mind.* New York: Harcourt.

Dayan, P., & Hinton, G. E. (1993). Feudal reinforcement learning. In C. L. Giles, S. J. Hanson & J. D. Cowan (Eds.), *Advances in Neural Information Processing Systems 5*. San Mateo, CA: Morgan Kaufmann.

Deacon, T., Eichenbaum, H., Rosenberg, P., & Eckmann, K. (1983). Afferent connections of the perirhinal cortex in the rat. *Journal of Comparative Neurology, 220*, 168-190.

Dietterich, T. G. (1998). The MAXQ method for hierarchical reinforcement learning. *Proceedings of the 15th International Conference on Machine Learning.*

Digney, B. L. (1996). Emergent hierarchical control structures: Learning reactive / hierarchical relationships in reinforcement environments. *Proceedings of the Fourth International Conference of Simulation of Adaptive Behavior.*

Dissanayake, G., Durrant-Whyte, H., & Bailey, T. (2000). A computationally efficient solution to the simultaneous localisation and map building (SLAM) problem. *Proceedings of the International Conference on Robotics and Automation*, San Fransicso.

Dyer, F. C. (1996). Spatial memory and navigation by honeybees on the scale of the foraging range. *The Journal of Experimental Biology, 199*, 147-154.

Eichenbaum, H. (1996). Is the rodent hippocampus just for place. *Current Opinion in Neurobiology, 6*, 187-195.

Eichenbaum, H., & Cohen, N. (1988). Representations in the hippocampus: What do hippocampal neurons encode. *Trends in Neurosciences, 11*:6, 244-248.

Eichenbaum, H., Otto, T., & Cohen, N. (1992). The hippocampus - What does it do? *Behavioral and Neural Biology, 57*, 2-36.

Etienne, A. (1987). The control of short-distance homing in the golden hamster. In P. Ellen & C. Thinus-Blanc (Eds.), *Cognitive Processes and Spatial Orientation in Animals and Man.* Boston: Martinus Nijhoff.

Fisher, R. (1936). The use of multiple measurements in taxonomic problems. *Annual Eugenics, 7*:Part II, 179-188.

Foster, D. J., Morris, R. G. M., & Dayan, P. (2000). A model of hippocampally dependent navigation, using the temporal difference learning rule. *Hippocampus, 10*:1, 1-16.

Gaskett, C., Fletcher, L., & Zelinsky, A. (2000). Reinforcement learning for a vision based mobile robot. *Proceedings of the International Conference on Intelligent Robots and Systems*, Takamatsu, Japan.

Gaussier, P., Joulain, C., Banquet, J. P., Lepretre, S., & Revel, A. (2000). The visual homing problem: An example of robotics/biology cross fertilization. *Robotics and Autonomous Systems, 30*, 155-180.

Gaussier, P., Revel, A., Banquet, J. P., & Babeau, V. (2002). From view cells to place cells to cognitive map learning: processing stages of the hippocampal system. *Biological Cybernetics, 86*, 15-28.

Georgopoulos, A. P., Kettner, R. E., & Schwartz, A. B. (1988). Primate motor cortex and free arm movements to visual targets in three-dimensional space. II. Coding of the direction of movement by a neuronal population. *Journal of Neuroscience, 9*, 2928-2937.

Goodridge, J. P., & Touretzky, D. S. (2000). Modelling attractor deformation in the rodent head-direction system. *The Journal of Neurophysiology, 83*, 3402-3410.

Guazzelli, A., Bota, M., & Arbib, M. A. (2001). Competitive hebbian learning and the hippocampal place cell system: Modelling the interaction of visual and path integration cues. *Hippocampus, 11*, 216-239.

Guazzelli, A., Corbacho, F. J., Bota, M., & Arbib, M. A. (1998). Affordances, Motivations, and the World Graph Theory. *Adaptive Behavior, 6*:3/4, 435-471.

Habib, M., & Sirigu, A. (1987). Pure topographical disorientation: A definition and anatomical basis. *Cortex, 23*, 73-85.

Harris, K. D., & Recce, M. (1997). Absolute localization for a mobile robot using place cells. *Robotics and Autonomous Systems, 22*, 393-406.

Hirtle, S., C., & Heidorn, P. B. (1993). The structure of cognitive maps: Representations and processes. In T. Gärling & R. G. Golledge (Eds.), *Behavior and Environment: Psychological and Geographical Approaches* (pp. 170-192). Amsterdam: North-Holland.

Hirtle, S., C., & Jonides, J. (1985). Evidence of heirarchies in cognitive maps. *Memory and Cognition, 13*:3, 208-217.

Jazwinski, A. (1970). *Stochastic processes and filtering theory*: Academic Press.

Kaelbling, L. P. (1993a). Hierarchical learning in stochastic domains: Preliminary results. *Proceedings of the International Conference on Machine Learning.*

Kaelbling, L. P. (1993b). Learning to achieve goals. *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambéry, France.

Kali, S., & Dayan, P. (2000). The involvement of recurrent connections in area CA3 in establishing the properties of place fields: a model. *The Journal of Neuroscience, 20*:19, 7463-7477.

Kim, J., & Seong, P. (1996). Experiments on orientation recovery and steering of an autonomous mobile robot using encoded magnetic compass disc. *IEEE Transactions on Instrumentramentation and Measurement, 45*:1, 271-274.

Kleeman, L. (1992). Optimal estimation of position and heading for mobile robots using ultrasonic beacons and dead-reckoning. *Proceedings of the International Conference on Robotics and Automation*, Nice, France.

Klopf, A. H. (1972). *Brain function and adaptive systems - A heterostatic study* (No. AFCRL-72-0164). Bedford, MA.: Air Force Cambridge Research Laboratories.

Koenig, S., & Simmons, R., G. (1993). Complexity analysis of real-time reinforcement learning. *Proceedings of the Eleventh National Conference on Aritificial Intelligence.*

Kohonen, T. (1995). *Self-Organizing Maps*. Berlin: Springer-Verlag.

Kudrimoti, H. S., McNaughton, B. L., Barnes, C., & Skaggs, W. E. (1995). Recent experience strengthens preexisting correlationships between hippocampal neurons during sleep. *Society for Neuroscience Abstracts, 21.*

Lee, D., & Recce, M. (1997). Quantitative evaluation of the exploration strategies of a mobile robot. *International Journal of Robotics Research, 16*:4, 413-447.

Lee, D. C. (1996). *Map-building and exploration strategies of simple sonar-equipped robot.* Cambridge: Cambridge University Press.

Leonardis, A., & Bischof, H. (1998). An efficient MDL-based construction of RBF networks. *Neural Networks, 11*, 963-973.

Leonhard, C. L., Stackman, R. W., & Taube, J. S. (1996). Head-direction cells recorded from the lateral mammillary nucleus in rats. *Society for Neuroscience Abstracts, 22*, 1873.

Mahadevan, S., & Connel, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence, 55*:2-3, 311-365.

Markus, E., Qin, Y., Leonard, B., Skaggs, E., McNaughton, B. L., & Barnes, C. (1995). Interactions between location and task affect the spatial and directional firing of hippocampal neurons. *Journal of Neuroscience, 15*, 7079-7094.

Marr, D. (1971). Simple memory: A theory for archicortex. *Philisophical transactions of the royal society of London, 262*, 23-81.

McNaughton, B. L., Barnes, C., Gerrard, J. L., Gothard, K., Jung, W., Knierim, J. J., et al. (1996). Deciphering the hippocampal polyglot: The hippocampus as a path integration system. *The Journal of Experimental Biology, 199*, 173-185.

McNaughton, B. L., Chen, L., & Markus, E. (1991). "Dead reckoning", landmark learning, and the sense of direction: A neurophysiological and computational hypothesis. *Journal of Cognitive Neuroscience, 3*, 190-202.

McNaughton, B. L., Markus, E., Wilson, M. A., & Knierim, J. J. (1993). Familiar landmarks can correct for cumulative error in the inertially based dead-reckoning system. *Society for Neuroscience Abstracts, 19*, 795.

Mittelstaedt, M., & Mittelstaedt, H. (1980). Homing by path integration in a mammal. *Naturwissenschaften, 67*, 566-567.

Moody, J., & Darken, C. J. (1989). Fast Learning in Networks of Locally-Tuned Processing Units. *Neural Computation, 1*, 281-294.

Morris, R. G. M. (1981). Spatial localization does not require the presence of local cues. *Learning and Motivation, 12*, 239-260.

Muller, R. U., & Kubie, J. L. (1987). The effects of changes in the environment on the spatial firing of hippocampal complex-spike neurons. *Journal of Neuroscience, 7*, 1951-1968.

Muller, R. U., & Kubie, J. L. (1989). The firing of hippocampal place cells predicts the future position of freely moving rats. *The Journal of Neuroscience, 9*:12, 4101-4110.

Muller, R. U., Kubie, J. L., Bostock, E. M., Taube, J. S., & Quirk, G. J. (1991). Spatial firing correlates of neurons in the hippocampal formation of freely

moving rats. In J. Paillard (Ed.), *Brain and Space* (pp. 296-333). New York: Oxford University Press.

Mura, A., & Feldon, J. (2003). Spatial learning in rats is impaired after degeneration of the nigrostriatal dopaminergic system. *Movement Disorders, 18*:8, 860-871.

Newman, D., Hettich, S., Blake, C., & Merz, C. (1998). *UCI repository of machine learning databases*, from http://www.ics.uci.edu/~mlearn/MLRepository.html

O'Keefe, J. (1976). Place units in the hippocampus of the freely moving rat. *Experimental Neurology, 51*, 78-109.

O'Keefe, J. (1989). Computations the hippocampus might perform. In L. Nadel, L. A. Cooper, P. Culicover & R. M. Harnish (Eds.), *Neural Connections, Mental Computation* (pp. 225-284). Cambridge: MIT Press.

O'Keefe, J. (1990). A computational model of the hippocampal cognitive map. In J. Storm-Mathisen, J. Zimmer & O. P. Ottersen (Eds.), *Understanding the Brain Through the Hippocampus: The Hippocampal Region as a Model for Studying Brain Structure and Function.* Amsterdam: Elsevier.

O'Keefe, J. (1991). The hippocampal cognitive map and navigational strategies. In J. Paillard (Ed.), *Brain and Space.* Oxford: Oxford University Press.

O'Keefe, J., & Dostrovsky, J. (1971). The hippocampus as a spatial map. Preliminary evidence from the unit activity in the freely-moving rat. *Brain Research, 34*, 171-175.

O'Keefe, J., & Nadel, L. (1978). *The hippocampus as a cognitive map.* Oxford: Clarendon Press.

O'Keefe, J., & Speakman, A. (1987). Single unit activity in the rat hippocampus during a spatial memory task. *Experimental Brain Research, 68*, 1-27.

Ollington, R. B., & Vamplew, P. W. (2003). Adaptive response function neurons. *Proceedings of the Second International Conference on Computational Intelligence, Robotics and Autonomous Systems*, Singapore.

Parr, R., & Russell, S. (1997). Reinforcement learning with hierarchies of machines. In M. I. Jordan, M. J. Kearns & S. A. Solla (Eds.), *Advances in Neural Information Processing systems* (Vol. 10): The MIT Press.

Peng, J., & Williams, R. J. (1996). Incremental multi-step Q-learning. *Machine Learning, 22*:1-3, 283-290.

Platt, J. (1991). A Resource-Allocating Network for Function Interpolation. *Neural Computation, 3*, 213-225.

Ranck, J. B. J. (1984). Head direction cells in the deep cell layer of dorsal presubiculum in freely moving rats. *Society for Neuroscience Abstracts, 10*, 599.

Recce, M., & Harris, K. D. (1996). Memory for places: A navigational model in support of Marr's theory of hippocampal function. *Hippocampus, 6*, 735-748.

Redish, A. D., Elga, A. N., & Touretzky, D. S. (1996). A coupled attractor model of the rodent head direction system. *Network: Computation in Neural Systems, 7*, 671-685.

Redish, A. D., & Touretzky, D. S. (1997). Cognitive maps beyond the hippocampus. *Hippocampus, 7*, 15-35.

Redish, A. D., & Touretzky, D. S. (1999). Separating hippocampal maps. In N. Burgess, J. Jeffrey & J. O'Keefe (Eds.), *The Hippocampal and Parietal Foundations of Spatial Cognition* (pp. 203-219). Oxford: Oxford University Press.

Rummery, G. A., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems* (No. CUED/F-INFENG/TR 166): Cambridge University Engineering Department.

Schultz, W., & Dickinson, A. (2000). Neuronal coding of prediction errors. *Annual Review of Neuroscience, 23*, 473-500.

Schultz, W., Tremblay, L., & Hollerman, J. R. (2000). Reward processing in primate orbitofrontal cortex and basal ganglia. *Cerebral Cortex, 10*, 272-283.

Sharp, P. E., Blair, H. T., Etkin, D., & Tzanetos, D. B. (1995). Influences of vestibular and visual motion information on the spatial firing patterns of hippocampal place cells. *Journal of Neuroscience, 15*:1, 173-189.

Sharp, P. E., & Green, C. (1994). Spatial correlates of firing patterns of single cells in the subiculum of the freely moving rat. *Journal of Neuroscience, 14*:4, 2339-2356.

Singh, S. P. (1992). Reinforcement learning with a hierarchy of abstract models. *Proceedings of the National Conference on Artificial Intelligence.*

Skaggs, E., Knierim, J. J., Kudrimoti, H., & McNaughton, B. L. (1995). A model of the neural basis of the rat's sense of direction. *Advances in Neural Information Processing Systems, 7*, 173-180.

Speakman, A., & O'Keefe, J. (1990). Hippocampal complex spike cells do not change their place fields if the goal is moved within a cue controlled environment. *European Journal of Neuroscience, 2*, 544-555.

Steele, R. J., & Morris, R. G. M. (1999). Delay-dependent impairment of a matching-to-place task with chronic and intrahippocampal infusion of the NMDA-Antagonist D- AP5. *Hippocampus, 9*:2, 118-136.

Suri, R. E. (2002). TD models of reward predictive responses in dopamine neurons. *Neural Networks, 15*, 523-533.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning, 3*, 9-44.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning*, San Mateo.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction.* Cambridge, MA: MIT Press.

Swanson, L. W., & Mogenson, G. J. (1981). Neural mechanisms for the functional coupling of autonomic, endocrine and somatomotor responses in adaptive behavior. *Brain Research Reviews, 3*, 1-34.

Taube, J. S., & Burton, H. L. (1995). Head direction cell activity monitored in a novel environment and during a cue conflict situation. *Journal of Neurophysiology, 74*, 1953-1971.

Taube, J. S., Muller, R. U., & Ranck, J. B. J. (1990). Head direction cells recorded from the postsubiculum in freely moving rats. *Journal of Neuroscience, 10*, 420-447.

Tolman, E. C. (1948). Cognitive maps in rats and men. *Psychological Review, 40*, 60-70.

Touretzky, D. S., Wan, H. S., & Redish, A. D. (1994). Neural representation of space in rats and robots. In J. M. Zurada & R. J. Marks (Eds.), *Computational Intelligence: Imitating Life*: IEEE Press.

Trullier, O., & Meyer, J. A. (1997). Place sequence learning for navigation. *Proceedings of the 7th international conference on artificial neural networks*, Lausanne, Switzerland.

Trullier, O., & Meyer, J. A. (1998). Animat navigation using a cognitive graph. *Proceedings of the Conference on Simulation of Adaptive Behaviour*, Zurich, Switzerland.

Wan, H. S., Touretzky, D. S., & Redish, A. D. (1994a). Computing goal locations from place codes. *Proceedings of the 16th annual conference of the cognitive science society*, Atlanta, GA.

Wan, H. S., Touretzky, D. S., & Redish, A. D. (1994b). Towards a computational theory of rat navigation. *Proceedings of the 1993 Connectionist Models Summer School.*

Watkins, C. J. C. H. (1989). *Learning from delayed rewards.* PhD. Dissertation Thesis, King's College, Cambridge.

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*:3-4, 279-292.

Wehner, R., & Räber, F. (1979). Visual spatial memory in desert ants *Cataglyphis bicolor. Experientia, 35.*

Wierling, M., & Schmidhuber, J. (1998). Fast online Q(λ). *Machine Learning, 33*, 105-115.

Wilson, M. A., & McNaughton, B. L. (1993). Dynamics of the hippocampal ensemble code for space. *Science, 261*, 1055-1058.

Wiltschko, R. (1997). The navigational system of birds. *Proceedings of the AISB workshop on "spatial reasoning in mobile robots and animals"*, Manchester.

Zhang, K. (1996). Representations of spatial orientation by the intrinsic dynamics of the head-direction ensemble: A theory. *The Journal of Neuroscience, 16*:6, 2112-2126.

Appendix A. Simulation

The simulated robot is not based on any real robot and is instead rather simplistic and abstract. Similarly the environment is also simplistic. Where not otherwise specified, all simulations were conducted in a spatially continuous environment with time discretised into 0.02s steps. For maximum flexibility, the simulation was created with Auran Jet[11].

Physical Parameters

The robot consisted of a square base with a width of 40cm. The movement rate was 1 m/s either forward or backward and the turn rate was 2 rad/s. The simulation did not model inertia and acceleration, and the robot was allowed to turn with no effect on forward or backward speed.

Sensors included 9 range sensors arranged as shown in Figure A.1. The sensor model (below) was designed to simulate in an abstract way the performance of a combination of sonar and IR sensors. The robot was also able to detect a collision with any part of the base.

[11] Auran Jet is a graphics engine that is available free for academic use. See www.auranjet.com

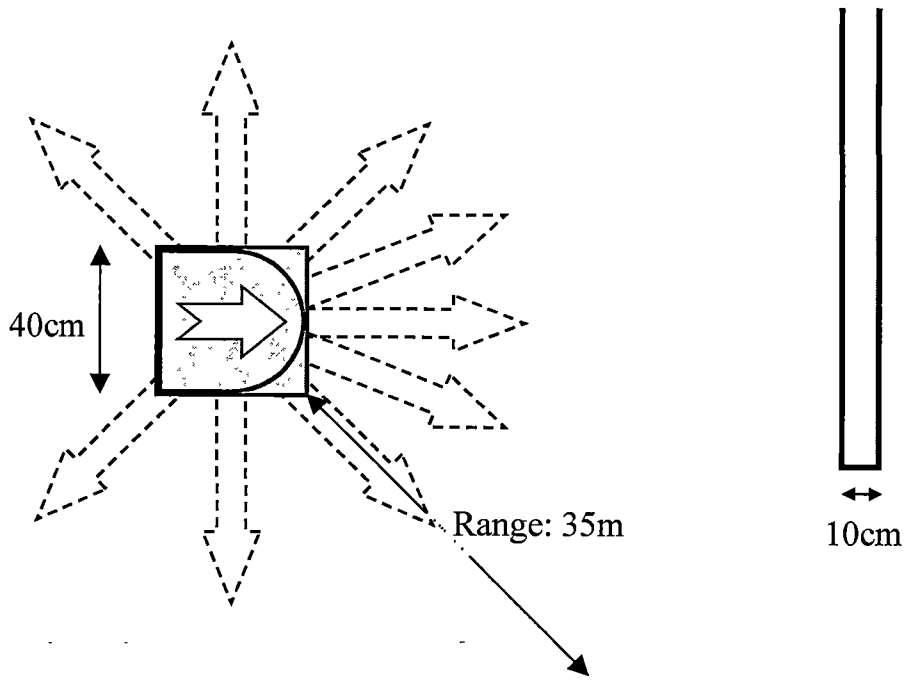

Figure A.1: Physical parameters of the simulation showing: robot size, wall thickness, sensor arrangement, and sensor range.

Range Sensors

The range sensor model was chosen to give sufficiently noisy readings without requiring excessive computational resources. For each sensor, one ray segment was created at each time step originating at the sensor location and extending for 35m in a random direction within 20 degrees of the sensor orientation. For each reading, a weighting was calculated based on the angle between the ray and the sensor facing using:

$$w(t) = \exp\left(-\frac{[\theta_r(t) - \theta_s(t)]^2}{\varpi^2} \right) \qquad \textbf{A.1}$$

where $w(t)$ is the weighting at time t, θ_r and θ_s are the directions of the ray segment and sensor respectively, and ω is the 'beam width' of the sensor. For all experiments, ω was set to10 degrees.

The sensor reading at time t was calculated using:

$$R(T) = \frac{\sum_{t=T-9}^{T} w(t)\, r(t)}{\sum_{t=T-9}^{T} w(t)} \qquad \textbf{A.2}$$

where $R(T)$ is the range returned by the sensor, and $r(t)$ is the collision range for the ray segment at time t.

This procedure produced noisy sensor data with many of the properties of real range sensors. For example, sensor readings are less accurate if the angle of incidence with the wall is acute, readings are less accurate near a corner, opening or wall edge, and readings are less accurate if the robot is moving.

Collision Sensing

The robot was able to detect collisions with any part of the base of the robot. These collisions generated a single collision signal with no indication of the area on the base with which the collision occurred. In addition, 'virtual' collisions were generated by the range sensor array to prevent collision where possible. If any of the five forward facing sensors indicated a range of less than 10cm, no further forward movement was allowed and a collision signal was generated. Similarly, if either of the backward facing sensors indicated a range less than 10cm, no further backward movement was allowed and a collision signal was generated. These virtual collisions provided a safety margin to protect the robot and the environment, and reduced the chance that the square robot would become wedged against a wall.

Odometry

Full odometric information was available to the robot. The current heading was considered completely accurate and no noise was added to this measurement in any experiments. The current velocity of the robot was also available, however noise was added to this measurement as specified in Chapter 6.

Appendix B. Symbols Used

The following tables list the symbols used for each section of the thesis. Also given is the default or initial values of parameters where applicable.

Table B.1: Adaptive Response Function Neurons (Section 5.1)

Symbol	Description	Initial/Default Value
$R(x)$	ARFN response for input x	-
r_e	Response of excitatory interneuron	-
r_i	Response of inhibitory interneuron	-
s_e	Synaptic efficiency of excitatory interneuron → ARFN connection	1.0
s_i	Synaptic efficiency of inhibitory interneuron → ARFN connection	1.0
t_e	Synaptic efficiency of bias input → excitatory interneuron connection	-
t_i	Synaptic efficiency of bias input → inhibitory interneuron connection	-
g_e	Synaptic efficiency of input → excitatory interneuron connection	-
g_i	Synaptic efficiency of input → inhibitory interneuron connection	-
η_t	Training rate for threshold	2.0
η_g	Training rate for gain	2.0
α	Equilibrium position for training	0.75
β	Additional equilibrium position for training gain	0.8

Table B.2: Place Cells (Chapter 6)

Symbol	Description	Initial/Default Value
VC	View cell output	-
PC	Place cell output	-
VI	View cell contribution to place cell firing	-
PI	Path integrator contribution to place cell firing	-
$\overline{p}'$	Position estimate	-
$\overline{p}$	Place field centre	-
σ	Gaussian width of path integration input	1.0
w_{ij}^{d}	Synaptic weight of the connection from view cell i to place cell j for direction d	-
η_v	Training weight for view cell input	0.5
η_p	Training weight for correcting position estimate	0.1
a	Multiplier for path integration input	3.6
b	Multiplier for view cell input	1.2
t	Sigmoid threshold for place cell activation	3.0

Table B.3: Temporal Difference Learning (Section 7.1)

Symbol	Description	Initial/Default Value
π	Policy	-
$V^{\pi}(s)$	Value of state s under policy π	-
r	reward	-
γ	Discounting factor	-
δ	Estimated error in value function	-
η	Training rate	-
η'	Training rate for actor	-
$p(s,a)$	Actor's preference for choosing action a from state s	-
ε	Probability of choosing a non-greedy action	-
$\mathcal{P}(s,a)$	Probability of choosing action a from state s	-
τ	Temperature for Boltzmann distribution	-
$Q(s,a)$	Value of choosing action a from state s	-
$e(s)$	Eligibility trace for state s	-
λ	Trace decay parameter	-
$\overline{\theta}$	Vector of parameters for function approximation	-
$\overline{\phi}_s$	Vector of measurements of state s	-

Table B.4: Collision Avoidance (Section 7.2)

Symbol	Description	Initial/Default Value
γ	Discounting factor	0.9
λ	Trace decay parameter	0.9
η	Training rate	0.1
τ	Temperature for Boltzmann distribution	0.1

Table B.5: Concurrent Q-learning (Chapter 8)

Symbol	Description	Initial/Default Value
$Q^{s^d}(s,a)$	Value of performing action a from state s, with respect to goal state s^d	-
$R^{s^d}(s,a)$	Expected return for reaching goal state s^d after performing action a from state s	-
r^{s^d}	Reward for reaching goal state s^d	-
γ	Discounting factor	0.95
λ	Trace decay parameter	0.95
η	Training rate	0.8
κ	Exploration bonus parameter	0.001
$n(s,a)$	Number of time steps since action a was performed from state s	10000
$T(s)$	Update threshold for state s	-

Table B.6: System Integration (Chapter 9)

Symbol	Description	Initial/Default Value
$\overline{x}_a$	Displacement vector for action a	-
$\overline{p}$	Place field centre	-
m	Multiplier for initialisation	8.0

Appendix C. View Dataset

Range 1	Range 2	View
5.98	15.47	0
6.2	15.47	0
5.31	16.98	0
5.7	16	0
5.57	17.18	0
7.33	17.89	0
8.4	18.61	0
5.32	15.04	0
5.64	18.59	0
6.89	18.5	0
6.11	15.44	0
5.41	17.05	0
5.43	15.72	0
11.15	20.26	0
5.36	15.78	0
6.59	16.43	0
5.91	16.47	0
7.37	18.33	0
9.91	19.62	0
5.67	15.89	0
5.18	16.87	0
13.6	19.78	0
7.74	19.38	0
6.11	15.62	0
5.94	16.31	0
6.21	16.77	0
5.84	17.31	0
5.92	15.86	0
5.62	17.66	0
8.39	19.3	0
9.77	19.18	0
12.04	19.22	0
5.71	17.18	0
8.08	18.48	0
5.24	15.95	0
9.68	17.72	0
5.56	16.6	0
8.97	19.64	0
5.73	15.51	0
5.07	15.97	0
8.33	17.31	0
6.52	17.86	0
5.52	15.5	0
6.06	15.91	0
7.97	19.13	0
5.28	16.7	0
5.22	16.9	0
6.09	16.21	0
6.65	18.51	0
9.26	19.83	0
11.74	19.37	0
5.58	18.04	0
12.83	19.57	0
5.3	16.6	0
5.45	15.07	0
5.26	14.77	0
5.57	16.63	0
7.95	18.09	0
6.17	17.62	0
7.5	18.36	0
5.95	17.33	0
4.94	15.42	0
8.92	19.15	0
9.68	18.46	0
11.26	19.91	0
6.5	18.45	0
5.44	17.07	0
6.01	15.85	0
8.03	18.94	0
5.56	15.4	0
5.24	15.96	0
5.46	15.41	0
5.25	14.56	0
9.05	20.08	0
5.37	15.84	0
7.27	16.47	0
5.64	17.52	0
9.91	18.74	0
5.69	16.37	0
11.13	19.05	0
12.47	20.19	0
8.41	18.9	0
5.95	15.47	0
7.93	18.66	0
13.56	20.61	0
5.64	17.79	0
5.72	16.43	0
5.53	15.27	0
5.57	17.04	0
5.49	16.18	0
6.05	15.61	0
8.76	18.82	0
7.14	18.88	0
5.43	15.48	0
10.1	19.73	0
5.52	15.59	0
5.6	15.8	0
5.78	15.51	0
11.08	20.08	0
7.34	17.39	0
5.97	14.27	1
5.77	14.43	1
5.81	13.71	1
5.74	14.16	1
5.68	14.27	1
6.44	13.99	1
6.51	13.86	1
5.5	13.72	1
5.84	14.59	1
6.12	13.9	1
5.64	14.36	1
5.71	14.13	1
5.92	13.91	1
7.11	11.87	1
5.96	13.87	1
6.41	13.62	1
5.81	14.38	1
6.36	14.05	1
6.92	13.07	1
5.99	14.11	1
5.63	13.94	1
7.58	10.19	1
6.29	14.3	1
5.31	14.39	1
5.64	14.38	1
6.27	13.94	1
5.68	14.49	1
5.83	14.29	1
5.91	14.4	1
6.4	14.06	1
6.74	12.92	1
7.49	11.12	1
5.83	14.37	1
6.57	14.11	1
5.46	13.81	1
7.18	13.72	1
5.55	14.16	1
6.55	13.6	1
6.25	13.66	1
5.71	13.69	1
6.92	13.84	1
6.3	14.46	1
5.33	13.94	1
5.09	14.39	1
6.38	14.19	1
5.85	13.77	1

5.59	13.97	1
6.4	14.14	1
6.08	14.02	1
6.53	13.14	1
7.32	11.4	1
5.88	14.35	1
7.6	10.52	1
5.47	13.97	1
5.31	13.83	1
5.83	13.64	1
5.5	14.17	1
6.66	14.15	1
6.11	14.08	1
6.49	14.3	1
6.16	14.4	1
5.49	13.51	1
6.87	14.64	1
7.14	14.15	1
7.21	11.8	1
6.17	14.47	1
5.66	14.16	1
5.15	14.35	1
6.61	14.56	1
6.02	13.95	1
5.39	13.82	1
6.18	13.83	1
5.25	13.6	1
6.54	13.43	1
6.02	13.69	1
6.66	13.57	1
5.99	14.37	1
6.93	13.09	1
5.3	14.21	1
7.23	11.92	1
7.56	10.77	1
6.65	14.35	1
5.67	14.25	1
6.55	14.3	1
7.65	10.13	1
5.94	14.32	1
6.13	13.97	1
5.99	13.91	1
5.88	13.63	1
5.6	14.03	1
5.23	14.34	1
6.71	14.19	1
6.24	14.29	1
5.86	13.87	1
6.92	12.8	1
5.93	13.96	1
5.17	14.05	1
6.31	13.79	1
7.16	11.95	1

6.52	13.82	1
12.18	11.42	2
11.96	11.61	2
12.55	10.62	2
12.09	11.22	2
12.28	11.22	2
13.3	10.85	2
13.64	10.65	2
11.76	10.84	2
12.7	11.45	2
13.21	10.71	2
11.84	11.54	2
12.33	11.08	2
12.27	10.98	2
14.31	11.46	2
12.33	10.92	2
12.99	10.57	2
12.2	11.43	2
13.32	10.88	2
14.04	11.28	2
12.32	11.19	2
12.27	10.88	2
14.83	10.95	2
13.47	11.07	2
11.54	11.54	2
12	11.45	2
12.84	10.9	2
12.25	11.46	2
12.11	11.39	2
12.58	11.32	2
13.64	10.86	2
13.98	10.72	2
14.58	10.96	2
12.39	11.33	2
13.54	10.94	2
11.91	10.84	2
14.04	10.57	2
12.05	11.16	2
13.78	11.08	2
12.6	10.71	2
12.19	10.7	2
13.64	10.73	2
12.98	11.37	2
11.64	11.04	2
11.39	11.51	2
13.52	10.98	2
12.49	10.71	2
12.23	10.91	2
12.77	11.18	2
13.13	10.83	2
13.85	11.01	2
14.48	10.94	2
12.67	11.23	2

14.74	11.07	2
12.03	10.94	2
11.57	10.96	2
12.05	10.79	2
12	11.17	2
13.5	11.01	2
12.87	10.97	2
13.35	11.15	2
12.74	11.35	2
11.9	10.55	2
13.8	11.46	2
14.03	11.02	2
14.36	11.34	2
13.01	11.33	2
12.28	11.11	2
11.44	11.47	2
13.53	11.4	2
12.28	11.06	2
11.84	10.84	2
12.47	10.92	2
11.46	10.77	2
13.8	11.29	2
12.45	10.71	2
13.26	10.51	2
12.63	11.3	2
14.04	10.7	2
11.74	11.24	2
14.35	10.84	2
14.67	11.55	2
13.64	11.17	2
11.9	11.4	2
13.49	11.12	2
14.86	11.58	2
12.67	11.22	2
12.62	10.96	2
12.24	11.04	2
12.67	10.53	2
12.04	11.06	2
11.47	11.49	2
13.74	11	2
13.27	11.11	2
12.17	10.96	2
14.07	11.25	2
12.24	11.04	2
11.53	11.12	2
12.62	10.86	2
14.32	11.45	2
13.29	10.7	2

Appendix D. Publications

Much of the work included in this thesis has been published, or is awaiting publication, in refereed conferences or journals.

Adaptive Response Function Neurons (Section 5.1)

Ollington, R. B., & Vamplew, P. W. (2003). Adaptive response function neurons. *Proceedings of the Second International Conference on Computational Intelligence, Robotics and Autonomous Systems*, Singapore.

Place Cell System (Sections 5.2 and Chapter 6)

Ollington, R. B., & Vamplew, P. W. (2004). Learning place cells from sonar data. *Proceedings of the International Conference on Artificial Intelligence in Science and Technology*, Hobart, Australia.

Concurrent Q-Learning (Section 8.2)

Ollington, R. B., & Vamplew, P. W. (2003). Concurrent Q-learning for autonomous mapping and navigation. *Proceedings of the Second International Conference on Computational Intelligence, Robotics and Autonomous Systems*, Singapore.

Ollington, R. B., & Vamplew, P. W. (2005). Concurrent Q-learning: Reinforcement learning for dynamic goals and environments. *International Journal of Intelligent Systems*, 20:10, 1037-1052.

Ollington, R. B., & Vamplew, P. W. (2004). Reducing the time complexity of goal-independent reinforcement learning. *Proceedings of the International Conference on Artificial Intelligence in Science and Technology*, Hobart, Australia.