Behaviour-Based Reinforcement Learning

George Dimitri Konidaris

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh

2003

Abstract

Although behaviour-based robotics has been successfully used to develop autonomous mobile robots up to a certain point, further progress may require the integration of a learning model into the behaviour-based framework. Reinforcement learning is a natural candidate for this because it seems well suited to the problems faced by autonomous agents. However, previous attempts to use reinforcement learning in behaviour-based mobile robots have been simple combinations of these two methodologies rather than full integrations, and have suffered from severe scaling problems that appear to make them infeasible. Furthermore, the implicit assumptions that form the basis of reinforcement learning theory were not developed with the problems faced by autonomous agents in complex environments in mind.

This dissertation introduces a model of reinforcement learning that is designed specifically for use in behaviour-based robots, taking the conditions faced by situated agents into account. The model layers a distributed and asynchronous reinforcement learning algorithm over a learned topological map and standard behavioural substrate to create a reinforcement learning complex. The topological map creates a small and task-relevant state space that aims to make reinforcement learning feasible, while the distributed and asynchronous nature of the model makes it compatible with behaviour-based design principles.

The model is then validated through an experiment that requires a mobile robot to perform puck foraging in three separate artificial arenas. The development of Dangerous Beans, a mobile robot that is capable of building a distributed topological map of its environment and performing reinforcement learning over it is described, along with the results of its use to test three control strategies (random decision making, a standard reinforcement learning algorithm layered on top of a topological map, and the full model developed in this dissertation) in the arenas. The results show that the model developed in this dissertation is able to learn rapidly in a real environment, and outperforms both the random strategy and the layered standard reinforcement learning algorithm. Following this, a discussion of the implications of these results is given, which suggests that situated learning and the integration of behaviour-based methods and layered learning models merit further study.

Acknowledgements

First, I would like to thank Gillian Hayes for agreeing to supervise me even though I was clearly crazy. Through her exceptional patience and insightful comments, Gillian allowed me to turn my fragmented and disorganised ideas into a coherent thesis, while keeping the scope of the project sane and focused. George Maistros, Chris Malcolm and John Hallam provided invaluable discussion, insight and advice, without which this research would have been much less interesting. This research would not have been possible at all without the equipment and cooperation of the Mobile Robot Group. In particular, whoever built the swivelling tether saved me an awful lot of trouble. I also owe a debt of gratitude to Irene Madison, Jane Rankin, Douglas Howie and Lizelle Bisschoff-Minnaar for arranging for me to stay in Forrest Hill for a little while after the Robot Lab moved, and thereby saving my bacon.

George Christelis was primarily responsible for my improbable survival over the past year. He, along with everyone else at Churchill House, has made this year more than memorable. Sarah Rauchas and Lex Holt were there, as always, forming nodes 1 and 2 in my social network. Steve McLean's excellent taste in music may quite possibly have kept me going during many long hours of shared toil in the Robot Lab.

My MSc at Edinburgh was funded by a Commonwealth Scholarship (ref. ZACS–2002–344) administered by the British Council, for which I am deeply grateful. My RSO, Alison Kanbi, has demonstrated superb skill at getting things done very quickly when necessary.

Finally, I am deeply indebted to my parents, Spiro and Marina, and my sister Tanya, for just about everything.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*George Dimitri Konidaris*)

for

Spiroandreas Sotirios Konidaris

6th August 1939 – 13th May 2003

deeply loved and sorely missed

Table of Contents

1 Introduction **1**

1.1 Introduction 1

1.2 Behaviour-Based Reinforcement Learning 2

1.3 Research Approach 2

1.4 Structure of the Dissertation 3

2 Background **4**

2.1 Introduction 4

2.2 Behaviour-Based Robotics 4

2.3 Reinforcement Learning 6

2.4 Distributed Learning Models 8

2.5 Layered Learning Models 9

2.6 Conclusion 10

3 Behaviour-Based Reinforcement Learning **11**

3.1 Introduction 11

3.2 Reinforcement Learning over Topological Maps 12

3.3 Distributed Reinforcement Learning 14

3.3.1 Temporal Difference Methods 15

3.3.2 Monte Carlo Methods 16
3.3.3 TD(λ) 16
3.4 Reinforcement Learning in Situated Agents 18
3.5 Examples 21
3.6 Summary 22

4 An Experiment: Puck Foraging in an Artificial Arena 23
4.1 Introduction 23
4.2 Overview 23
4.3 Evaluation 27
4.4 Arena 1: Testing and Simple Puck Finding 27
4.5 Arena 2: A Hostile Environment 29
4.6 Arena 3: A Far Away Puck 30
4.7 Summary 31

5 Distributed Map Learning in an Artificial Arena 33
5.1 Introduction 33
5.2 The Environment 33
5.2.1 Physical Characteristics 34
5.2.2 Arena Configurations 34
5.3 The Robot 34
5.3.1 Hardware 36
5.3.2 Software Interface 36
5.3.3 Behavioural Substrate 37
5.3.4 Landmark Detection 40
5.3.5 Map Building 42

5.4 Summary 45

6 Distributed Reinforcement Learning in an Artificial Arena 47

6.1 Introduction 47

6.2 Implementation 48

6.2.1 Obtaining Reward 48

6.2.2 Circadian Events 50

6.2.3 Making Choices 50

6.2.4 Place and Transition Values 51

6.2.5 Data Capture 53

6.2.6 Modifications to the Original Map-Building System 53

6.3 Results 55

6.3.1 The First Arena 55

6.3.2 The Second Arena 59

6.3.3 The Third Arena 62

6.3.4 Convergence 65

6.3.5 Summary 66

6.4 Conclusion 66

7 Discussion 67

7.1 Introduction 67

7.2 Significance 67

7.3 Limitations 68

7.4 Implications 69

7.4.1 Situated Reinforcement Learning 69

7.4.2 Planning Behaviour 70

7.4.3 Layered Learning 70
7.4.4 Emergent Representations 71
7.5 Conclusion 73

8 Conclusion **74**
8.1 Introduction 74
8.2 Contribution 74
8.3 Significance 74
8.4 Future Work 75
8.5 Conclusion 76

Bibliography **77**

List of Figures

3.1 Graphical and Tabular Action Value Representations 15

3.2 Rewards Received Along a Transition Path using TD(0) and TD(λ) 17

4.1 Potential Actions for a Wall Following Robot 25

4.2 The First Experimental Arena 28

4.3 The Second Experimental Arena 29

4.4 The Third Experimental Arena 31

5.1 The Three Arenas 34

5.2 Dangerous Beans 35

5.3 Dangerous Beans: Behavioural Structure (Map Building) 35

5.4 Dangerous Beans: Sensory Configuration 36

5.5 Front Sensor Obstacle Avoidance Thresholds 39

5.6 Uncorrected and Angle-Corrected Dead-Reckoning Maps 42

5.7 Angle-Corrected and Fully-Corrected Dead-Reckoning Maps 44

5.8 A Distributed Topological Map of the First Arena 46

6.1 Dangerous Beans: Behavioural Structure (Reinforcement Learning) 48

6.2 Sample Pixel Array Images 49

6.3 Average Puck Reward over Time: The First Arena 55

6.4 Learned and Random Routes to the Puck in the First Arena 56
6.5 Average Home Reward over Time: The First Arena 57
6.6 Learned and Random Routes Home in the First Arena 57
6.7 Preferred Transitions Maps for the First Arena 58
6.8 Average Puck Reward over Time: The Second Arena 59
6.9 Learned Puck Finding Behaviour in the Second Arena 60
6.10 Preferred Puck Transitions Maps for the Second Arena after the Eighth Cycle . 61
6.11 Average Home Reward over Time: The Second Arena 61
6.12 Average Puck Reward over Time: The Third Arena 63
6.13 Learned Puck Routes in the Third Arena 63
6.14 Preferred Puck Transitions Maps for the Third Arena 64
6.15 Average Home Reward over Time: The Third Arena 64
6.16 ATD Average Action Value Changes Over Time 65

Chapter 1

Introduction

1.1 Introduction

Any theory of intelligence must account for the wide spectrum of learning mechanisms displayed by insects, animals and humans. Although some aspects of an autonomous agent can be evolved or directly engineered, other elements of behaviour require learning because they involve knowledge that can only be gained by the agent itself, or that may change in unpredictable ways over its lifetime. Although behaviour-based robotics has had some success as a basis for the development of intelligent, autonomous robots, the way in which learning fits into the behaviour-based framework is not yet well understood.

Reinforcement learning is well suited to the kinds of problems faced by the current generation of behaviour-based robots. It provides goal-directed learning without requiring an external teacher, handles environments that are not deterministic and rewards that require multiple steps to obtain, and has a well-developed theoretical framework (Sutton and Barto, 1998). Sutton (1990) has even argued that the problem facing an autonomous agent *is* the reinforcement learning problem. Because of this, several researchers have included reinforcement learning in their robots. However, these have either involved using reinforcement learning over the robot's entire sensor space, and have thus suffered from scaling problems (e.g., Mahadevan and Connell (1992)) or did not involve real robots at all, but attempted to retrofit existing architectures for use in potential robotic applications (e.g., Sutton (1990)).

This dissertation introduces a model of reinforcement learning that is designed specifically for

use in behaviour-based robots, motivated by the behaviour-based emphasis on layered competencies and distributed control, and argues that the use of reinforcement learning in situated agents creates a different set of concerns than those emphasised by the reinforcement learning literature. It outlines an experiment aimed at evaluating this model, details the development of Dangerous Beans, a mobile robot that implements the model, and presents the results of its use in the experiment. The results demonstrate that the model is capable of rapid learning, resulting in behavioural benefits in real time on a real robot.

1.2 Behaviour-Based Reinforcement Learning

This dissertation is about the fusion of the behaviour-based style of robot control architecture and reinforcement learning. Reinforcement learning is usually considered in isolation, and conceived of as a single process operating on a private central data structure. When it has been added into a robot control system, it has usually been via a reinforcement learning module that contains a standard implementation of reinforcement learning as a single behavioural module. Although this is in a sense behaviour-based, one of the central arguments put forward by this dissertation, and the motivation behind the model presented in it, is that this represents a simple combination of these two methods rather than a true integration.

For truly behaviour-based reinforcement learning, the representation should be the control system, and the control system should function as the representation. A behaviour-based architecture must provide a basis upon which reinforcement learning can be layered, rather than just one into which it can be inserted. Similarly, reinforcement learning should not be isolated from the rest of the system, but rather integrated into an already existing distributed control system. Such a model would map naturally to a neural implementation. The model presented in this dissertation is the result of one attempt at such an integration.

1.3 Research Approach

The research approach adopted in this dissertation is that of *synthetic modelling* (Pfeifer and Scheier, 1999). This entails the development of a model and the synthesis of an instantiation of that model that can be used for empirical testing and analysis. The approach taken therefore involved the design of an experiment where the model developed here could be evaluated. This

experiment required the construction of an artificial environment and the development of a mobile robot designed to operate in it while making use of the model.

The aim of the experiment was twofold. First, the construction of an actual robot system aimed to prove that the model is feasible, in terms of both computational requirements and design effort. Second, it allowed for the empirical evaluation of the benefits provided by the model, and a qualitative evaluation of the resulting behaviour.

Although this approach is inherently limited in that it only evaluates one individual instantiation of the model for one particular task, it provided solid data that led to a positive evaluation of the model, and established a good basis for further research.

1.4 Structure of the Dissertation

The following chapter briefly outlines behaviour-based robotics and reinforcement learning, and covers related research in both fields. It examines previous work on layered learning models and distributed representations, and attempts to make the reasoning behind the development of the reinforcement learning model developed in this dissertation clear. Chapter 3 introduces the new reinforcement learning model, discusses the concerns raised when using reinforcement learning in situated agents, and provides some examples of where the model could be applied.

Chapter 4 presents an experiment based on one of these examples, which requires a mobile robot in an artificial arena to learn to find a food puck, explore, and return home, using reinforcement learning over a distributed map of the arena. Chapter 5 details the construction of three artificial arenas and the development of Dangerous Beans, a robot that is capable of performing distributed map building in them. Following this, Chapter 6 describes the addition of a reinforcement learning layer to the robot and presents the results of its use in the experiment. The results show that the model presented in Chapter 3 is feasible, and capable of learning in real time.

Chapter 7 provides a discussion of the significance, limitations and implications of the model and the experiment, and Chapter 8 concludes by summarising the contribution and significance of this dissertation and outlining potential future work.

Chapter 2

Background

2.1 Introduction

Behaviour-based robotics and reinforcement learning are both well developed fields with rich bodies of literature documenting a wide range of research. This chapter presents a brief overview of the basic ideas and literature related to these fields, paying particular attention to areas that are relevant to the model and experiment presented in this dissertation, and covers some of the work that combines both fields. It aims to show that behaviour-based robotics and reinforcement learning are a natural combination, but that no previous approach has been fully successful in merging the two.

The next section provides an overview of behaviour-based robotics and its principles, followed by a brief outline of reinforcement learning theory, along with some key examples of its application to robotics. An overview of the distributed learning models found in the literature is then given, and recent research into layered learning models is covered. The final section concludes.

2.2 Behaviour-Based Robotics

Behaviour-based robotics is centred around the idea that the best way to study intelligence is through the development of mobile robots (Brooks, 1991a). Prior to this, nearly all artificial intelligence research consisted of the focused study of one small aspect of behaviour, with

the idea that by decomposing intelligence it would be easier to understand each of its parts. Brooks (1991a), however, claimed that such decompositions may be misleading because they are typically based on introspection, which is a notoriously unreliable method of psychological analysis. Furthermore, researchers studying individual aspects of intelligence may lose sight of the feasibility and interface constraints they should be adhering to.

Behaviour-based robotics thus emphasises the construction of complete, functional agents that must exist in the real world. Agents that exist within and interact with such complex environments in real time are known as *situated agents*. Situated agents must confront the issues of real time control and the complexity of the world directly – they must behave in the real world in real time.

One of the consequences of this change in emphasis has been the development of a different set of research concerns than those traditionally considered important in artificial intelligence. Behaviour-based robotics emphasises the use of distributed, parallel and primarily reactive control processes, the emergence of complex behaviour through the interaction of these processes with each other and the environment, cheap computation, and the construction of agents through the layered addition of complete and functional behavioural levels. The last point is important because it facilitates the incremental construction of mobile robots and explicitly seeks to mimic the evolutionary development of behavioural complexity. The behaviour-based approach is capable of developing agents that demonstrate surprisingly complex behaviour (e.g., Braitenberg (1984)), and has developed into a significant field of research with three major textbooks (Arkin, 1998; Pfeifer and Scheier, 1999; Murphy, 2000).

Although behaviour-based robotics is starting to become widely accepted as a methodology for developing mobile robots, and has been able to produce working robots for a variety of interesting problems, it has difficulty developing systems that display a level of intelligence beyond that of insects. This is partly because the majority of present research focusses on *hybrid architectures*, which combine ideas from traditional artificial intelligence and behaviour-based robotics (Bryson, 2002), and partly because there has been no thorough investigation into the issues of behaviour-based representations and the integration of learning methods into behaviour-based systems.

Brooks (1991b) argued that the traditional approach to machine learning has produced very few learning models that are applicable to the problems faced by situated agents. The research presented in this dissertation aims to develop just such a learning model.

2.3 Reinforcement Learning

The reinforcement learning problem is the problem of learning to maximise a numerical reward signal over time in a given environment (Sutton and Barto, 1998). The reward signal is the only feedback obtained from the environment, and thus reinforcement learning falls somewhere between unsupervised learning (where no signal is given at all) and supervised learning (where a signal indicating the correct action is given) (Mitchell, 1997).

More specifically, given a set of states[1] S and a set of actions A, reinforcement learning involves either learning the values of each $s \in S$ (the state value prediction problem) or the value of each state-action pair (s, a), where $s \in S$ and $a \in A$ (the control problem) (Sutton and Barto, 1998). For most tasks, these values can only be estimated given experience of the reward received at each state or from each state-action pair through interaction with the environment. This estimate is usually achieved by building a table that contains an element for each desired value and using a reinforcement learning method to estimate the value of each element.

The three primary solution methods employed to solve reinforcement learning problems are Dynamic Programming, Monte Carlo estimation and Temporal Difference methods. Dynamic Programming methods make use of an a priori environmental model to achieve an exact solution to the reinforcement learning problem without requiring any interaction with the environment, but are only applicable in cases where such a model is available. Monte Carlo methods are only applicable to episodic tasks[2] and estimate each state or state-action pair based on the total reward received from its first occurrence in an episode until the episode's termination. Finally, Temporal Difference methods estimate the value of a particular state or state-action pair using its current value, the reward received when it is active (in the case of a state) or taken (in the case of state-action pair) and the value of the following state or state-action pair. These methods *bootstrap* because they calculate state or state-action pair values using the values of other states or state-action pairs (Sutton and Barto, 1998). One important Temporal Difference algorithm, TD(λ), combines the advantages of Monte Carlo methods and Temporal Difference methods. A more comprehensive treatment of the wide range of reinforcement learning methods and their theoretical properties cannot be given here; however, an an excellent overview of the field is given in Sutton and Barto (1998).

[1] These states are required to be Markov states – each individual state must contain sufficient information to determine the optimal action for the agent without any other knowledge of the agent's history.

[2] Episodic tasks are tasks that must consist of only a finite number of state transitions – in other words, they must be guaranteed to end.

Reinforcement learning is attractive to researchers in robotics because it provides a principled way to build value driven agents – agents whose actions are guided by a set of internal drives (Pfeifer and Scheier, 1999). Furthermore, it has a sound theoretical basis, can allow for the principled integration of a priori knowledge, handles stochastic environments and rewards that take multiple steps to obtain, and is intuitively appealing. Because it has so many attractive properties, several researchers have added reinforcement learning capabilities to their robots. An early example of this was the development of Obelix (Mahadevan and Connell, 1992), a robot that learned to push boxes by reinforcement. Although Obelix was able to learn in real time, it required a hand-discretised state space and the use of statistical clustering in order to do so, even though the robot's sensor space was only eighteen bits.

The straightforward application of reinforcement learning to robot applications invariably leads to such problems. Since such models typically use the robot's sensor space directly as the reinforcement learning state space, they suffer from serious performance and scaling problems – a robot with just sixteen bits of sensor space has over sixty five thousand states. Convergence in such a large state space will take a reinforcement learning algorithm a very long time. One solution to this problem is the use of simulators, in which very long training times are acceptable (e.g., Toombs *et al.* (1998)), but such agents cannot be considered situated. These problems have led some researchers to develop hierarchical reinforcement learning methods that aim to make learning more tractable through the use of varying levels of detail (e.g., Digney (1998) and Morén (1998)), and others to use complex statistical methods to speed up learning (e.g., Smart and Kaelbling (2000)). Another approach is the use of a function approximation method to approximate the value table (Sutton and Barto, 1998). However, this introduces the additional the problem of selecting a good approximation method for the task at hand, and does not retain many of the theoretical guarantees known to apply to tabular reinforcement methods.

The fundamental problem with using reinforcement learning methods in mobile robots is that they were not developed with the problems faced by situated agents in mind. Matarić (1994) gives an important criticism of the direct application of reinforcement learning to behaviour-based robotics which reflects the idea that the implicit assumptions made in the reinforcement learning literature need to be reexamined in the context of situated agents. Further discussion of the differences between the concerns emphasised by the reinforcement learning literature and those posed by situated agents is given in Chapter 3.

It is clear from this discussion given above that the use of reinforcement learning methods

directly over a robot's sensor space is not feasible, and that a good solution to the problem of applying reinforcement learning in situated agents has yet to be found.

2.4 Distributed Learning Models

The behaviour-based emphasis on distributed and parallel control processes implies that any learning model included in a behaviour-based system should also be distributed. This section discusses some of the distributed learning models found in the literature.

Although neural network learning models have always been inherently distributed (Pfeifer and Scheier, 1999), supervised learning algorithms such as back-propagation (Mitchell, 1997) are not appropriate for use in autonomous agents. There are, however, two classes of neural learning mechanisms that could be useful for behaviour-based robots – Hebbian learning models and self-organising feature maps.

In Hebbian learning, when two neurons are active at the same time, an excitatory connection between them is established if one does not exist, or the existing one is strengthened if it already does. This method is completely distributed in the sense that no central control is required whatsoever provided each neuron can determine when it should be active. Braitenberg (1984) uses this learning method almost exclusively to develop autonomous agents that display quite complex behaviour. One useful variation on Hebbian learning is Value-based Hebbian learning, where the excitatory connection is strengthened only when some given stimulus (e.g., the smell of food) is present (Pfeifer and Scheier, 1999). This is aimed at creating connections only in cases that are significant according to a value system in which the stimulus is important[3]. Hebbian methods are primarily useful for associative learning tasks.

Self-organising feature maps (SOFMs) are sets of interconnected neurons that organise themselves to match the important features of an input space while preserving its topological properties. The best known SOFM is the Kohonen network (Kohonen, 1989), which consists of a fixed-size network with fixed connectivity that adapts itself to match the structure inherent in its inputs, but more dynamic types of SOFMs include Growing Neural Gas (Fritzke, 1995), which creates networks with dynamic size and connectivity, and Grow When Required (Marsland *et al.*, 2002), which does the same but adds nodes according to accumulated error. SOFMs are

[3]Reinforcement learning can be considered a form of Value-based Hebbian learning where the excitatory connection between two nodes is strengthened when they are active after each other, instead of at the same time.

primarily useful for automatically learning the structure of an input space, and are potentially of considerable importance since topological maps appear to be ubiquitous in natural systems (Ferrel, 1996).

Only two major behaviour-based systems have been built using distributed learning models. The first involves the learning of activation conditions for a set of behaviours that must coordinate to produce emergent walking behaviour on a six-legged robot (Maes and Brooks, 1990). Although the results produced by this algorithm were impressive, it is highly task-specific and not likely to be useful elsewhere.

The second instance of a distributed learning behaviour-based robot is given in Matarić and Brooks (1990), and is of particular relevance to this dissertation. Matarić and Brooks (1990) detail the development of a robot called Toto, that was capable of wandering around an office environment and learning a distributed topological map of it inspired by the role of "place cells" in the rat hippocampus. This map was made up of independent behaviours, each of which became active and attempted to suppress the others when the robot was near the landmark it corresponded to. Each landmark behaviour also maintained a a list of the other landmark behaviours that had previously followed it, and spread expectation to them, thereby increasing their sensitivity. Because the behaviours were all active in parallel, the distributed map provided constant time localisation and linear time path planning using spreading expectation (Matarić, 1990).

Matarić (1990) can be considered the first known instance of an emergent data structure, and this dissertation can be considered an extension of the line of research that first appeared in it.

2.5 Layered Learning Models

The layered learning methodology was introduced by Stone (2000), and was intended to deal with problems where learning a direct mapping from input to output is not feasible, and where a hierarchical task decomposition is given. The method involves using machine learning at several layers in an agent's control system, with each layer's learning directly affecting that of subsequent layers. One learning model may affect another through the provision of its training examples[4] or through the construction of its input or output features (Stone, 2000).

[4]Although the use of training examples is not appropriate in situated learning, a learning model in a situated agent could for example bias the kinds of learning opportunities another model receives.

The layered learning method has generated impressive results – simulated soccer playing robots developed using it have twice won RoboCup, the robotic soccer championship (Stone, 2000). However, despite its obvious promise, layered learning has not yet been applied to a fully situated agent. Most of its implementations have been in simulation (Stone, 2000; Stone and Veloso, 2000; Whiteson and Stone, 2003), where training times can be much longer than those that would be acceptable for physical robots. Furthermore, the original stipulation that one layer should finish learning before another can start (Stone, 2000) is not realistic in situated environments (although recent research has allowed them to learn concurrently (Whiteson and Stone, 2003)).

Another relevant application of the layered learning approach is the use Kohonen networks to discretise continuous input and output spaces in order to make them suitable for reinforcement learning algorithms in Smith (2002). Although the results presented in Smith (2002) are promising, the algorithm is hampered by the requirement that the Kohonen map's parameters must be determined experimentally, and by the network's fixed dimensionality. The latter problem could potentially be solved through the use of more dynamic self-organising networks (e.g., Grow When Required (Marsland *et al.*, 2002)), but the former problem implies that the learning model is only feasible when it is task-specific.

Since Kohonen networks are topological maps, the model presented in Smith (2002) is in some respects similar to the one presented in this dissertation. However, it is not intended for use in situated agents and does not take the important interaction between the state and action spaces into account, as will be explained in Chapter 3

2.6 Conclusion

Although reinforcement learning has the potential to greatly broaden the behavioural scope of behaviour-based robots, current implementations have not been fully integrated into the behaviour-based methodology, and suffer from serious problems in terms of learning speed and scalability. By combining the layered learning, parallel control and distributed learning methods described in this chapter, the following chapter develops a reinforcement learning model that attempts to fully integrate behaviour-based robotics and reinforcement learning in order to produce real robots that are capable of rapid learning.

Chapter 3

Behaviour-Based Reinforcement Learning

3.1 Introduction

This chapter develops a model of reinforcement learning in autonomous agents motivated by the behaviour-based emphasis on distributed control and layered behavioural competencies. The model aims to broaden the scope of behaviour-based systems to include tasks that the ability to learn by reinforcement makes feasible.

The model presented here is novel for three reasons. First, it embeds a reinforcement learning layer in a distributed topological map that serves as a state space, instead of using the robot's sensor space directly. Second, rather than using one central control process and a single action-value or state-value table, the reinforcement learning functionality is distributed, requiring no central control and spreading the reinforcement table over the distributed map. This in effect creates a *reinforcement learning complex* that is emergent in the sense that it is contained in none of the individual processes or nodes, but results from their interaction with each other. Finally, because of the distributed nature of the map, the model makes use of asynchronous updates, where temporal difference updates at each node in the map take place in parallel and all the time, rather than only once at a state when a transition from that state takes place. This allows for the convergence (or near-convergence) of the reinforcement complex in situated agents where the time taken by an update is very small relative to the time required to perform

the actions that make up a single transition in the map.

The model is developed as follows. The next section introduces the idea of layering reinforcement learning over a topological map, and is followed by an explanation of how reinforcement learning can be performed in a distributed fashion, embedded within a distributed topological map. A discussion is then given of the ways in which reinforcement learning in situated agents creates a different set of concerns than those of standard reinforcement learning approaches, and introduces Asynchronous Temporal Difference (ATD) learning. This is followed by three examples of how this model could be applied to difficult learning problems. The final section summarises.

3.2 Reinforcement Learning over Topological Maps

Reinforcement learning models used for mobile robots typically use the robot's sensor space directly as the reinforcement learning state space. This results in a very large, redundant state space where states only have the Markov property for reactive tasks. The size of the state space means that it is difficult to achieve good coverage of it in a reasonable amount of time (forcing the use of function approximation and generalisation techniques) and that convergence of the state or state-action value table will take a very long time. Because of this, there are very few known examples of behaviour-based robots developing useful skills in real time using reinforcement learning.

The model proposed here makes use of an intermediate layer that learns a topological map of the sensor space, over which reinforcement learning takes place. A topological map is defined here as a graph with a set of nodes N and a set of edges E such that each $n \in N$ represents a distinct state in the problem space and an edge $e = < n_i, n_j >$ indicates that state n_i is topologically adjacent to state n_j *with respect to the behavioural capabilities of the agent*. This means that if there is an edge connecting these two nodes, then the activation of some simple behavioural sequence in the robot's control system will (perhaps with some probability) move the robot from state n_i to the state n_j.

The use of a topological map as a state space for a reinforcement learning algorithm has three major advantages over using the robot's sensor space directly. First, it discards irrelevant sensor input and results in a much smaller and task-relevant state space. This state space will scale well with the addition of new sensory capabilities to the robot because it is task dependent rather

than sensor dependent – new sensors will increase the robot's ability to distinguish between states, or perhaps present a slightly richer set of states, but will not introduce an immediate combinatorial explosion. Further, the topological map is not likely to be densely connected, making value propagation over the state space faster. Reinforcement learning over a topological map is therefore much more likely to be tractable than reinforcement learning over a large state space.

Second, the map's connectivity allows for a smaller action space, where actions are movements between nodes in the map rather than raw motor commands. Since such actions will naturally correspond to behaviours in a behaviour-based robot, the reinforcement learning layer can be added on top of an existing behaviour-based system without greatly disturbing the existing architecture, and without requiring exclusive control of the robot's effectors.

Finally, the states in the topological space are much more likely to be Markov states than raw (or even pre-processed) sensor snapshots. This extends the range of reinforcement learning methods for behaviour based robotics to tasks that are not strictly reactive, and removes the need for generalisation, because similar but distinct states are no longer likely to have similar values.

An important aspect of the proposed model is the interaction of an assumed behavioural substrate, the topological map, and the reinforcement learning algorithm. The behavioural substrate makes learning the topological map feasible, and provides the discrete actions which allow for movement between nodes on the topological map. Rather than simply using the topological map as a discretisation, the interaction between the topological map and the behavioural substrate is sufficient for it to be considered a grounded representation. The topological map, in turn, makes the use of reinforcement learning feasible. Finally, the strategy used for exploration at the reinforcement learning level may influence the way that the topological map develops, since learning at the topological map level continues at the same time as learning at the reinforcement learning level.

This emphasis on interaction differentiates the model presented so far from previous attempts to layer reinforcement learning over other learning models. For example, Smith (2002) introduced a similar model, where a Kohonen network (Kohonen, 1989) is used to discretise continuous input and output spaces, and reinforcement learning is performed over the resulting discretisation. However, since that model is not intended for use in an autonomous agent, it uses two separate maps (for the purposes of discretisation only), and does not take the rela-

tionship between the state and action space into account. Furthermore, because Smith uses a Kohonen map, the number of nodes in the map does not change, although their position does.

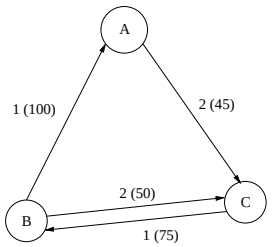
The major implication of the reliance on a topological mapping level is that it requires a tractably maintainable map that provides a good abstraction for the task at hand and can be grounded in the real world. Although there are methods (for example, Grow When Required (Marsland *et al.*, 2002)) that can automatically create and update topological maps for a given state space with no other knowledge, these methods are likely to be of use only when nothing is known about the sensor space at all. In real robot systems, a priori knowledge about the relative importance of different sensor inputs, the relationships between different sensors, and the types of sensor states that are important for the task at hand are all likely to be crucial for the development of a topological map learning layer. In such cases the development of that layer may be a harder problem than the application of the reinforcement learning model developed here on top of it.

3.3 Distributed Reinforcement Learning

Reinforcement learning is typically applied using a single control process that updates a single state or state-action value table. However, because a topological map is a dynamic structure, and because behaviour-based principles require distributed representation and parallel computation where possible, a distributed structure updated by many processes in parallel would be preferable. Since topological maps can easily be built in a distributed fashion (e.g., Matarić (1990)), this section describes how the reinforcement learning update equations can be adapted to run in a distributed fashion over a distributed map.

When performing reinforcement learning over a topological map (with nodes representing states and edges representing actions), we can view the learning as taking place over the nodes and transitions of the graph rather than over a table with a row for each node, or a row for each node and a column for each action type. Figure 3.1 illustrates the graphical and tabular representations for a simple example action-value set with states A, B and C, and action types 1 and 2, with the action-values given in brackets in the graph.

In a distributed topological map, each node would have its own process which would be responsible for detecting when the node it corresponds to should be active, and when a transition from it to another node has occurred. This allows each node in the map to maintain its own list



	1	**2**
A	–	45
B	100	50
C	75	–

Figure 3.1: Graphical and Tabular Action Value Representations

of transitions.

In order to add reinforcement learning to the topological map, each process must be augmented with code to perform an update over either the state or state-action spaces, using only information that can be obtained from the current node, one of the nodes directly connected to it, and a reward signal which must be globally available. Since almost all reinforcement learning update methods are intrinsically local, they require very little modification. The following sections consider each update type in turn, and briefly describe how they can be implemented in such a distributed system. All of the update equations given below are from Sutton and Barto (1998).

3.3.1 Temporal Difference Methods

Temporal difference methods are the easiest methods to implement in a distributed fashion because temporal update equations involve only local terms. The standard one-step temporal difference update equation (known as TD(0)) is:

$$V(s_t) = V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$$

where α and γ are global constants, $V(s_t)$ is the value of the active state at time t, and r_t is the reward received at time t. In order to implement this update equation, each node's process has only to note when it becomes active, the value of the state that is active immediately after it ceases to be active, and the reward received during the transition. It should also record the behaviours activated to cause the transition, and establish a link between the two nodes if one is not already present.

The update equation used for state-action value updates (known as Sarsa) is a slightly more difficult case. The Sarsa equation is:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

where $Q(s_t, a_t)$ is the value of taking action a_t from state s_t. This requires a node to have access to the value of the state-action pair following it, as well as the reward obtained between activations. Although this is still relatively local in that each node requires only the information from a single other node, one way to reduce the information sharing required would be to perform the update in terms of a state value, since state values can be computed using state-action values. The update equation would then be:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - Q(s_t, a_t))$$

where $V(s_{t+1})$ can either be the expected value of s_{t+1} calculated probabilistically, or simply the expected value of the action with the highest value from that state. The latter case is equivalent to Q-learning since then $V(s_t) = \max_{a_t} Q(s_t, a_t)$.

3.3.2 Monte Carlo Methods

The Monte Carlo estimate for the value of a state s is given by:

$$V(s) = \frac{R_s}{n_s}$$

where s has been visited during n_s episodes, and R_s is the sum of the returns[1] received after each first visit to state s. Although this method does not use information from any of the relevant node's neighbours in the distributed map, provided each node can determine when it has been activated and deactivated, when the end of an episode has been reached, and the reward received over time, obtaining a Monte Carlo state value estimate in a distributed fashion is straightforward. State-action values can be obtained similarly.

3.3.3 TD(λ)

The TD(λ) reinforcement learning algorithm provides a combination of the TD(0) temporal-difference update equation and Monte-Carlo estimation. Rather than performing the temporal difference backup using just the value of the state following the current state, TD(λ) uses a weighted average of all of the following states. The parameter λ affects the extent of the algorithm's similarity to TD(0) (where $\lambda = 0$) and Monte Carlo methods (which can be obtained

[1] Return is the total reward received from some time t until the end of an episode. Monte Carlo methods are therefore only applicable to episodic tasks.

when $\lambda = 1$). TD(λ) methods are usually implemented using an eligibility trace. Each node has its own eligibility trace value which records the extent to which the current reward and state value should affect the node's value. Each node's eligibility trace is decreased over time, increasing only when the node it corresponds to is active. Provided each node has access to the current reward and the value of the current state, the distributed implementation of eligibility traces (and thus TD(λ)) is straightforward.

The primary advantage of using TD(λ) is that it allows for faster learning when many steps are required to obtain a reward (Sutton and Barto, 1998). This is because on the initial discovery of the reward state, all of the states on the path receive some reward when TD(λ) is used, whereas only the state immediately before the reward state obtains a reward with a one-step method (such as TD(0)). This is illustrated in Figure 3.2, which is based on a similar illustration in Sutton and Barto (1998).

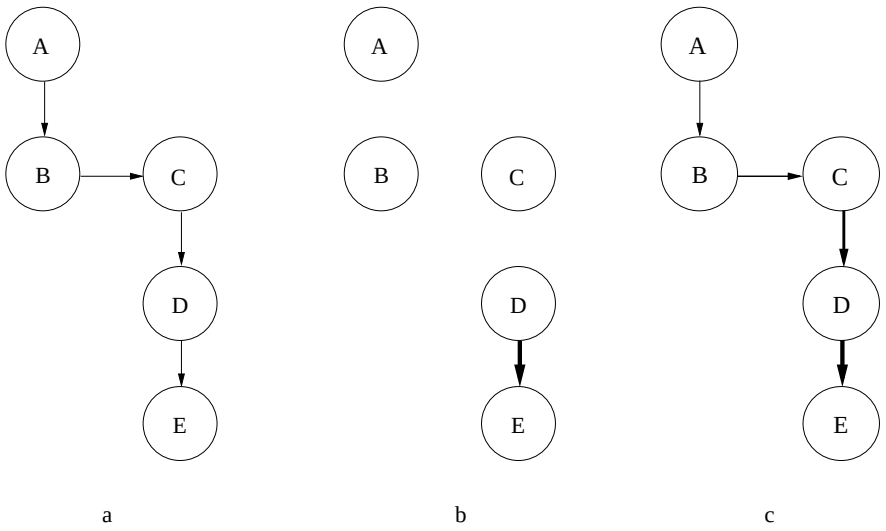


Figure 3.2: Rewards Received Along a Transition Path using TD(0) and TD(λ)

In Figure 3.2, all states except state E have zero reward. When the path indicated in *a* is taken using TD(0) and E is reached, only the transition from D to E receives some reward (indicated by the thick arrow from D to E in *b*). However, when TD(λ) is used, all of the transitions along the path obtain some reward as depicted in *c*, although the reward level drops the further away from E the transition is, as indicated by the thinner arrows.

In general, when using TD(0) at least a further $n-1$ transitions must be made after first finding the reward state before a path of length n can be constructed from the start point to the reward

state. However, the Asynchronous Temporal Difference algorithm introduced in the following section removes this limitation.

3.4 Reinforcement Learning in Situated Agents

Although Reinforcement Learning has a strong theoretical basis, it was not developed with the problems facing situated agents in mind. Instead, most of reinforcement learning theory assumes abstract state and action spaces and emphasises asymptotic convergence and optimality guarantees. The use of reinforcement learning in situated agents that must learn quickly in the real world leads to a different set of issues:

- **Situated agents are living a life** (Agre and Chapman, 1990). A situated agent has more than one task, and more than one concern. For example, a soda-can collecting robot must also avoid obstacles, navigate, and recharge its batteries when necessary. A reinforcement learning system will make up only one part of the robot's control system, and may have to share control with other reinforcement learning systems. It therefore cannot assume that it will always have control of the robot, and that the actions it requests will always be taken. One of the implications of this is that a situated agent will likely have many sensors and many motor behaviours, not all of which will be relevant to the task at hand. Another is that the robot may experience transitions on its topological map not governed by its reinforcement learning policy, and should be able to make use of these experience anyway. Therefore, on-policy learning methods may not be appropriate.
- **Asymptotic exploration is too slow**. Although the use of ε-greedy action selection methods provide asymptotic coverage of the state space, they are not likely to do so in a reasonable amount of time. Furthermore, they require the occasional completely irrational action from the agent, which may be dangerous or costly. The use of optimistic initial values is likely to result in better behaviour over a smaller time frame. Alternatively, some form of "exploration drive" could be built into the agent separately. The inclusion of such a drive with a low priority has the added advantage of allowing the robot to explore only when it has free time. However, in situations with large state spaces, the robot may have to be satisfied with a suboptimal solution.
- **Transitions do not take a uniform time**. The use of a global γ parameter to model the devaluation of reward over time is not appropriate in a real environment. When

performing an update over a transition, an estimate of the time taken by the transition is already available, since the agent has experienced the transition at least once. However, future states should not lose value simply because in some abstract way they are in the future; rather they should lose value because time and energy must be expended to get to them. This loss should be factored into the reward function for each transition. Similarly, the use of a global λ parameter for TD(λ) is no longer appropriate because λ-based erosion of eligibility traces involves an implicit assumption that all transitions take the same amount of time.

- **Rewards are not received discreetly with respect to actions and states**. In some situations, an agent may receive a reward while moving from one state to another, and in others it may receive a reward sometime during its presence at a particular state. The characteristics of the task must be taken into careful consideration when choosing a model.

- **Transitions take a long time**. In the case of a situated agent, the time taken to complete a transition and the time spent at each state are likely to be very much longer than the time required to perform a single update. Furthermore, since the reinforcement learning is being performed in a distributed fashion with one process per node, in principle all of the nodes can perform an update in parallel in the time it would take a single update to occur in a serial implementation. Although this can only be simulated on a single-processor computer, it would actually happen in parallel on a device capable of parallel computation (such as a brain, or a piece of neurally inspired hardware). This implies that many updates may take place between transitions.

- **Other learning models may be required in conjunction with reinforcement learning**. Situated learning is required to provide useful results quickly, and the use of reinforcement learning by itself may take too long. Fortunately, the reinforcement learning model given here provides an underlying representation that is well suited to the inclusion of other learning models through the modification of the reward function, the seeding of the initial state or state-action values, or the selection of motor behaviours and sensory inputs.

The penultimate point implies that rather than performing updates once per transition, a situated agent should be performing them all the time, over all nodes in parallel. However, in order to do this, the reliance of the update equations on the concept of “the transition just experienced”

must be removed. Therefore, it makes sense to use the experiences the agent has obtained so far to provide state and state-action value estimates, and to use these instead of the reward and state values actually experienced. Experienced values are thus used to create a model from which state or action-state value estimates are taken. For example, each node could update its state-action values using the following equation:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha(r_{s,a} + \gamma E_{s,a}\{V(s_{t+1})\} - Q_t(s,a))$$

where $r_{s,a}$ could be estimated as the average of all rewards received after executing a at s, and $E_{s,a}\{V(s_{t+1})\}$ would be the expected state value obtained after the execution of action a from state s. The expected state value could be the weighted (by observed probability) average of the states visited immediately after s, with each state value taken as the value of the maximum action available from that state. The γ parameter could be set to 1, or to a transition-specific decay value. This update equation would then be executed *all the time*. This method, Asynchronous Temporal Difference (ATD) learning, is used (with the above equation) in Chapter 6 to develop an asynchronous reinforcement learning robot.

This model draws from three ideas in the Reinforcement Learning literature. Like Dynamic Programming (Sutton and Barto, 1998), ATD learning uses a model to perform what is termed a *full backup*, which uses the expected value of a state or state-action pair rather than a sample backup, which uses a sampled value. However, unlike Dynamic Programming, the model used is derived from experience with the environment, and is not given a priori. This is similar to batch-updating, where the update rules from a given transition experience are repeatedly applied, but differs in that it does not simply repeat previous episodes, but uses a model of the environment to generate value estimates, and performs backups over all the nodes in the distributed map. Finally, ATD learning is similar to the Dyna model proposed by Sutton (1990) in that a model of the environment is built and used for reinforcement. It differs in that the updates occur in parallel and all the time, use the model to generate expected rather than sample state and state-action values (although Dyna could easily be adapted to allow this) and does so for all state-action pairs in each state rather than a single one.

Ideally, the use of asynchronous updates leads to the convergence of the values in the reinforcement learning complex *between transitions* so that at each transition the agent is behaving as best it can, given the information that it has obtained. This means that a situated agent using this model will make the best choices possible given its experiences, and make the most use of the limited information it has obtained from the environment.

3.5 Examples

One situation where the reinforcement learning model proposed here would be useful is the case of a rat learning to find a piece of food in a maze. The nodes in the topological map would correspond to landmarks in the maze, with a connection between two of them indicating that the rat is able to go from the first to the second. Reinforcement would be based on the rat finding the food. One potential application of additional learning models would be the use of associative learning to modify the reinforcement function so that locations where the smell of cheese is present receive some fraction of the reward received for finding the food. The experiment proposed in the following chapter is based on this example.

Another application could be the use of reinforcement learning in the development of simple motor skills for a robot with many actuators. For example, given a set of motor behaviours and joint angle sensors, a robot with a mechanised arm could use the reinforcement learning model proposed here to learn to reach out and touch an object. In this case the joint angle sensors in conjunction with the motor behaviours would provide the basis for the topological map, where nodes would be significant joint angle configurations and edges between them would indicate that movement between two configurations is possible with some short sequence of motor behaviours. In this case a self-organising map (such as Grow When Required (Marsland *et al.*, 2002) with appropriate input space scaling factors) could be used to create the topological map. The robot would receive a reward for touching the object, and the use of visual feedback could provide a heuristic that could modify the reinforcement function. Although this task seems easy given visual feedback it might be possible for the robot to learn to do it quickly and with little visual feedback or error with the aid of reinforcement learning.

Reinforcement learning could also be used for more complex motor coordination tasks, such as changing gear in a car with a manual transmission. This requires a fairly difficult sequence of actions, using a leg to engage the clutch and an arm to change gear. The map here would again be based on the joint angle sensors for the arm and leg and a set of motor behaviours. The use of social imitation could serve to seed the initial state values in order to make the task tractable – this is a fairly difficult learning task that takes humans a fair amount of time, effort and instruction to learn to perform smoothly.

In all three examples, the selection of the relevant sensors and motor behaviours is crucial. For example, it would be very difficult for a robot to learn to touch an object with its arm when all

of the internal joint sensors in its entire body were considered as input to a topological map, even though some of them might be relevant (it would be difficult to touch the ball while facing away from it, for example). The use of other learning models may aid in the selection of the relevant sensors and motor behaviours. In addition, other learning models may be useful in speeding up the learning – or may in fact be required to make learning feasible.

3.6 Summary

This chapter has presented a model of reinforcement learning for autonomous agents motivated by the behaviour-based emphasis on layered competencies and distributed control. The model is intended to produce behavioural benefits in real time when used in a real robot. It is novel for three reasons. First, it performs reinforcement learning over a learned topological map, rather than directly over the robot's sensor space. This aims to make learning feasible through the use of a small, relevant space tailored for the task at hand. Second, reinforcement learning is performed in a distributed fashion, resulting in a reinforcement learning complex embedded in a distributed topological map rather than a single state or state-action value table updated by a single control process. This allows for a dynamic structure that could potentially be updated in parallel with the use of parallel hardware. Finally, in order to take advantage of this parallelism, and the fact that situated agents will take much longer to make a transition than to perform an update, learning takes place all the time. Experiences are used to update an internal distributed model of the environment which is used as a basis for reinforcement learning, rather than being used in the reinforcement learning directly.

The use of reinforcement learning in situated agents raises a different set of concerns than those emphasised by the reinforcement learning literature. Asymptotic guarantees and convergence in the limit are not useful in real environments. Situated agents must learn quickly, performing as best they can given the limited information they can obtain from the environment in a finite time.

To illustrate the potential applications of the model introduced in this chapter, three example systems where reinforcement learning could provide behavioural benefits have been outlined. The first of these examples (the example of a rat in a maze) forms the basis of an experiment presented in the following chapter designed to test whether or not the model presented here can be feasibly implemented on a real robot and provide behavioural benefits in real time.

Chapter 4

An Experiment: Puck Foraging in an Artificial Arena

4.1 Introduction

This chapter presents an experiment designed to determine whether or not the model presented in Chapter 3 can be feasibly implemented on a mobile robot, and can provide behavioural benefits in real time. The experiment aims to augment the distributed map-building model used by Matarić (1990) with the reinforcement learning model introduced in Chapter 3, and show that this can produce complex, goal-directed and path-planning behaviour in an agent that performs puck foraging in an artificial arena.

The next section provides an overview of the experimental design, followed by an outline of the evaluation methods used in the experiment. Three arena configurations, each designed to highlight different aspects of the model, are then presented and discussed, and the final section summarises.

4.2 Overview

The experiment outlined here is intended as an abstraction of the rat in a maze example given in Chapter 3, which is itself an abstraction of the kinds of tasks commonly faced by foraging animals. It models an agent living in a static environment with obstacles that it must avoid, but

that it can use as landmarks for the purposes of navigation. The agent is driven by three internal needs – the need to find food, the need to explore its environment, and the need to return home. These needs are in turn activated and deactivated by a circadian cycle.

A mobile robot is placed in an artificial arena containing orthogonal walls (henceforth referred to as "vertical" and "horizontal" walls, since this is how they appear in figures) and one or more food pucks. The robot must start with no prior knowledge about the layout of the arena, and must navigate it for ten cycles. Each cycle is made up of three phases:

1. **Foraging**, where the robot should attempt to find a food puck (there may be more than one) in as short a time as possible. As soon as the robot has found a food puck, it switches to the exploration phase of the cycle. If it cannot find a food puck within a given period of time (the cycle length), it must skip the exploration phase and move directly to the homing phase.

2. **Exploration**, where the robot should explore areas of the arena that are relatively unexplored for the remainder of the cycle length. When the remainder of the cycle length has passed, the robot switches to the homing phase. The exploration phase is intended to allow the robot to build up a more accurate and complete map of its environment, if it has time after finding food.

3. **Homing**, where the robot must return to the area where it was first started. This is intended as analogous to nightfall, where the agent must return to its nest or home to sleep. As soon as it has done this, the robot moves to the next cycle and begins foraging again.

During its run, the robot is required to follow walls, and decide which action to take at the end of each one. The robot is restricted to one of three types of actions – turn right, left, or go straight – on either end of the wall, giving six actions in total. The robot therefore has to follow the wall until it reaches the desired end, and execute the desired action. However, the robot may not turn away from the wall, so only four of the six potential actions are available for walls. When the robot is in a corridor, it may choose from all six. Figure 4.1 shows the available actions for horizontal walls and corridors (the vertical case is similar).

Not all actions are possible in all cases – for example, if the left side of the corridor continues on to become a left wall while the right side does not, the robot may not turn left at that end. Therefore, the robot must determine whether or not an action is possible, and as far as possible

Figure 4.1: Potential Actions for a Wall Following Robot

avoid attempting illegal actions.

The state space for the reinforcement learning function will therefore be the set of landmarks present in the distributed map, and the action space will be the set of legal actions at each landmark. A transition would then consist of a landmark, a turn taken there, and the first landmark detected after the execution of the turn, with the transition being been considered completed once that landmark has been detected.

In order to implement the robot's internal drives, and to provide a useful metric for comparing various models, each drive is given an internal reward function. The following equations were used for the foraging, exploration, and homing rewards respectively:

$$f(x) = \begin{cases} 200 & \text{when a food puck is in view} \\ -1 & \text{otherwise} \end{cases}$$

$$e(x) = \begin{cases} 200 - 200\frac{n_t}{n_{ave}} & n_{ave} > 0 \\ 200 & \text{otherwise} \end{cases}$$

$$h(x) = \begin{cases} 200 & \text{when the robot is "home"} \\ -1 & \text{otherwise} \end{cases}$$

where n_t is the number of times that the transition just executed has just been taken in total, and n_{ave} is the average number of previous executions over all the transitions in the map. The exploration reward function should be executed once per transition, while the other should be

executed with a constant time delay set so that the robot would receive a penalty of at or near 200 for failing to find the puck before the end of a cycle.

The robot should at each choice point aim to maximise the sum over time of the value of the internal reinforcement function corresponding to the current cyclic phase. Although it is possible to try to minimise all three over time, for simplicity, and in order to avoid the difficulties of arbitrating between different reinforcement learning systems in a single agent (see Humphrys (1996) for a discussion), only the reward function pertaining to the current phase was considered at any given time.

In order to validate the model presented in Chapter 3, three decision models were used for the experiment:

1. **Random Movement**, where the robot chooses an action at random from the set of currently legal actions. This agent builds an internal distributed map of the arena but uses it only to determine which actions are legal. This model corresponds to the strategy typically employed in behaviour-based systems.

2. **Synchronous Reinforcement Learning**, where the robot builds an internal distributed map of the arena and uses a standard single-step temporal difference learning algorithm over it. This model embodies the application of traditional reinforcement learning techniques using a topological map as the state space, as described in Chapter 3.

3. **Asynchronous Reinforcement Learning**, where the robot builds an internal distributed map of the arena but uses the asynchronous temporal difference (ATD) model introduced in Chapter 3 over it, and constitutes the first implementation of a fully behaviour-based reinforcement learning robot.

The experiment facilitated the direct comparison of the three models in terms of performance (expressed by the internal reward functions) and behavioural characteristics. It also allowed for the evaluation of each model's performance over three different types of reinforcement functions. The reward function for obtaining the puck in some cases had multiple reward points (because there was more the one puck), and each was required to be discovered, whereas the homing reward function had exactly one reward point, which was known from the beginning of the first run because the robot started there. Finally, the exploration reward had many reward points of differing values, and was non-stationary because it changed rapidly over time – visiting a particular node immediately dropped its exploration reward value because the number

of times it had been visited increased. However, since the exploration phase was of varying length and did not occur in every cycle, the results obtained for it for each model could not be directly compared.

4.3 Evaluation

The learning models proposed in Chapter 3 are required to be evaluated both quantitatively and qualitatively. A quantitative analysis is appropriate for comparing reinforcement learning models directly against each other, but a qualitative analysis is required in order to assess the resulting behaviour and its apparent complexity.

To evaluate each model quantitatively, the reward values of each internal drive over time, averaged over a number of runs, were to be directly compared, thereby using the reward functions directly as performance metrics. Also, the average change of state value function in the map over time was to be considered, along with the transitions that added reward to the map, in order to examine the convergence of the asynchronous reinforcement learning model.

In order to evaluate each model qualitatively, more data was to be used from the same set of runs. First, the distributed map learned by the robot and the action values for it were to be visualised. In this way, the robot's internal map and action values could be examined. Further, recordings were to be made of the robot's movements so that specific instances can be replayed and examined, with a view to considering instances where choices had to be made and the robot's response to them. Internal data from the state of the robot could be used to obtain further information about the reasons behind the choices made.

Although this data was captured for all of the runs for a particular model, the presentation and analysis of only a few representative and informative examples is given in Chapter 6 in order to avoid repetition.

4.4 Arena 1: Testing and Simple Puck Finding

The first arena had a dual purpose: it provided a testing platform for the development of a robot that could build distributed maps, and it served as a simple problem instance where the reinforcement learning models could be compared to a random agent. The configuration of the

first arena is depicted in Figure 4.2. As with all of the arenas, the robot starts in the bottom left corner, facing right.

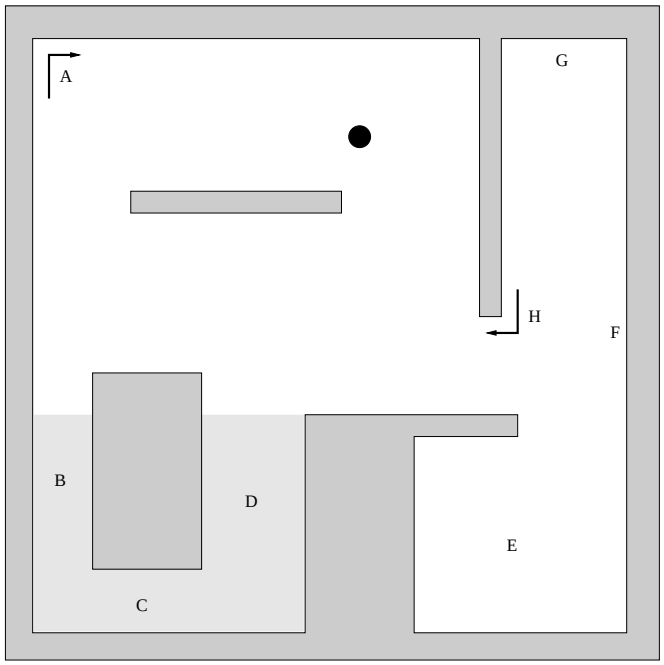


Figure 4.2: The First Experimental Arena

The first arena configuration was designed specifically to contain all of the important (and likely difficult) features that could be present in an arena. The lower left corner had two corridors (B and C) with a corner between them, which was predicted to be useful for testing corridor following and cornering behaviour, and the lower right corner had a box-shaped section (E) which was intended to cause maximum sensor noise. The arena also contained instances of landmarks with the minimum length the robot should be able to detect (e.g., C and G) and one landmark with the maximum length the robot should be able to detect (F). Finally, the arena also contained a section that was too just too wide to be considered a corridor (D) in order to test corridor detection. As with all of the arenas, the lightly shaded indicates the region used as the robot's home area, and the black circle indicates the position of the puck.

The aim of the experiments to be conducted in the first arena was to verify that the proposed models worked – that they were able to learn to find food pucks, explore, and return home as quickly as possible. The problem instance embodied by the first arena was made deliberately friendly toward the reinforcement learning agents. The area on the right (containing the box

marked as E as well as the region near F and G) is a kind of trap – once entered, it can only be escaped through the turn marked as H. Thus, a random agent is likely to spend a great deal of time in the trap, whereas a reinforcement learning agent should be able to learn to escape via H fairly quickly. The transition indicated by A is the only transition that leads to a puck, and is a only few transitions away from the home area. Since the robot is likely to return home using corridor B (often after turning at H), and thereby start trying to find the puck again from B, only two consecutive correct transitions are required in order to find the puck again (moving from B towards A and then taking A). Thus, both reinforcement learning models were expected to be able to learn to find the puck relatively quickly, and because the number of steps to each goal was short, the results for the two learning models were expected to be similar.

4.5 Arena 2: A Hostile Environment

The second arena configuration was intended to provide an environment that was relatively hostile to reinforcement learning agents and relatively friendly toward random agents, and is depicted in Figure 4.3.

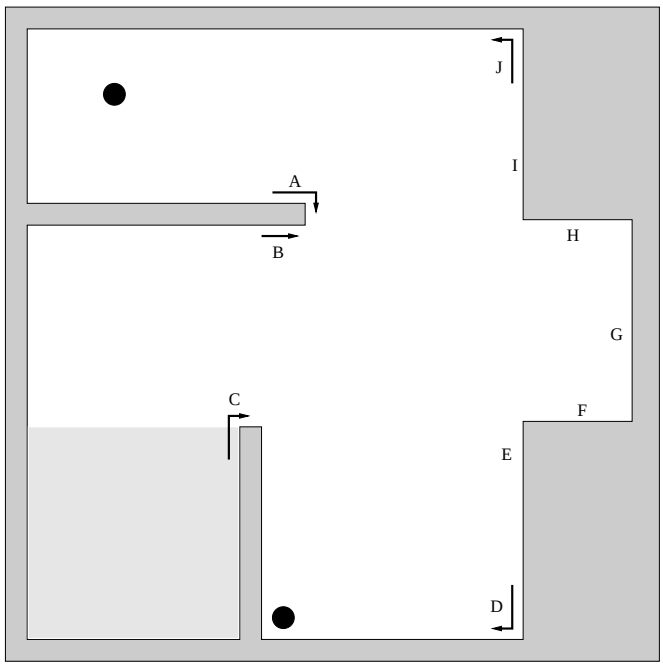


Figure 4.3: The Second Experimental Arena

Two features of the arena make it difficult for reinforcement learning agents. First, the quickest transitions out of the home area were noisy. Because the robot is unlikely to leave the wall at an angle that is exactly straight, turn B could lead to any of the walls labelled I, H or G. Similarly, the turn labelled C could lead to any of the walls labelled E, F or G. The noisy transitions were intended to test the robustness of the reinforcement learning methods in the presence of actions with uncertain results.

In addition, one of the pucks was taken away at the end of the fifth cycle. For the reinforcement learning agents, this was to be the last puck found during a foraging phase, or either if none had been seen so far. For the random agent, either puck could have been removed. This was intended to test how quickly the reinforcement learning agents were able to learn to go directly to the other puck after their previous first choice had been removed. Here, the transitions labelled J and D (and potentially the one labelled A), were intended to lead to a puck reward.

Finally, the second arena was intended to be a simple arena where a random agent was fairly likely to stumble across both the pucks and the home area. The arena was therefore intended to be useful in determining whether or not the reinforcement learning agents could do better than a random agent when faced with a hostile environment, and to evaluate the differences in recovery time between the standard and asynchronous reinforcement learning models after the first puck has been removed.

4.6 Arena 3: A Far Away Puck

The third arena was designed to test the ability of the reinforcement learning agents to discover a puck placed many transitions away in a complex environment, and to then be able to quickly find the puck again from home. The third arena configuration is depicted in Figure 4.4.

The third arena was to be the most complex task environment the robot faced. In order to successfully find the puck from home, the robot was required to make at least five consecutive correct transitions, following the path from A to E in Figure 4.4 (there are other paths, some of which may take less time, but none of which have fewer transitions). Transitions E and F are puck scoring transitions, although F cannot be reached without having seen the puck first because there is no other way to get to the ledge below it, and thus would not be expected to be on the robot's path to the puck. Returning home may require an even longer path. The path lengths required were intended to highlight the difference between the synchronous and

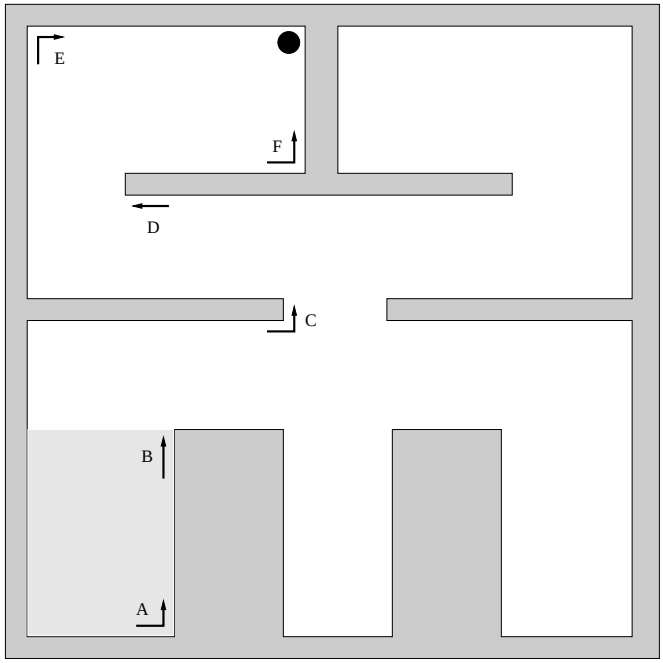


Figure 4.4: The Third Experimental Arena

asynchronous reinforcement learning models, where the agent using the asynchronous model was expected to be able to learn to find the path almost immediately after finding the puck for the first time, whereas the agent using the synchronous model was expected to take longer.

4.7 Summary

This chapter has presented an experiment that aims to test whether or not the reinforcement learning model introduced in Chapter 3 can be implemented in a real robot and provide behavioural benefits in real time. The task was intended as an abstraction of the rat in a maze example given in Chapter 3: a robot was placed in an artificial arena and required to find a food puck, explore, and then return to its home area, with its current activity regulated by a circadian cycle.

Since the particular arena used in any one instance of the task determined its difficulty, three arena configurations were presented, each intended to provide insight into a different aspect of the models' performance. The first arena was intended to demonstrate that the reinforcement

learning models are better than random choice in a fairly benign environment, while the second arena was intended to show that they are better even in a relatively hostile environment, and to compare the recovery times of the two reinforcement learning models after the removal of a puck. The final arena was designed to allow for the comparison of the two reinforcement learning models in a difficult environment where longer sequences of actions were required to be learned.

The following chapter describes the construction of the three arenas and the development of a robot that is capable of learning distributed maps. Chapter 6 then describes the addition of the two reinforcement learning models and the random decision models to the robot's control system, and presents the results of their use in the experiment outlined here.

Chapter 5

Distributed Map Learning in an Artificial Arena

5.1 Introduction

The experimental design outlined in Chapter 4 requires significant preparation: it requires the construction of a physical arena, and the development of a behaviour-based robot capable of distributed map building in that arena. This chapter describes the physical characteristics of the arena that was constructed for the experiment, and the hardware, interface to, and behavioural layers of Dangerous Beans, the robot that was used to implement map building behaviour in it.

5.2 The Environment

As required, the environment constructed for the experiment resembled a small maze or arena, with no objects other than pucks and horizontal and vertical walls present. Because an off-the-shelf robot was used (see section 5.3), the environment had to be carefully engineered to ensure a good fit between the robot's capabilities and the arena, in order to facilitate the development of the desired behaviour (Pfeifer and Scheier, 1999).

5.2.1 Physical Characteristics

The arena was built on a wooden base measuring 90cm by 90cm, with the walls constructed using pieces of styrofoam and insulation tape, and covered with sheets of white cardboard secured with drawing pins. The same type of cardboard was used to round off sharp internal corners. The use of the cardboard served to provide a smooth response for the infra-red sensors used, and the rounded corners simplified cornering and wall-following behaviour. The other materials were chosen because they were readily available.

Three white wooden cylinders from the lab were used as food pucks, with a strip of black insulation tape marking them for easy visual detection.

5.2.2 Arena Configurations

Each of the three configurations introduced in Chapter 4 was built on the same platform and with the same pieces of styrofoam, requiring the experiments performed on them to be completed sequentially. The three configurations are shown in figure 5.1. Note the presence of the food pucks in each.

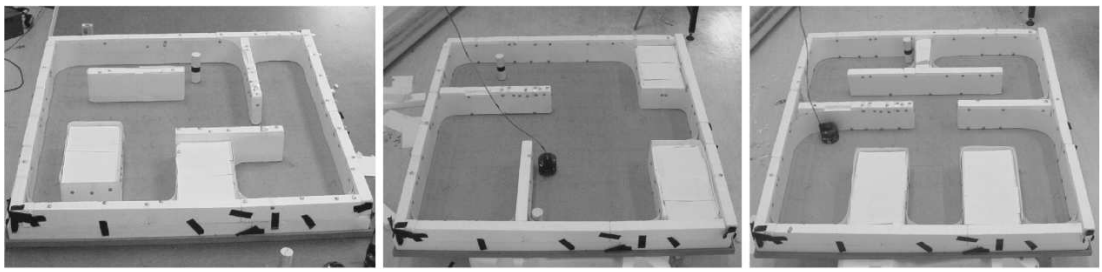


Figure 5.1: The Three Arenas

5.3 The Robot

This section describes the hardware, software, and behavioural levels of Dangerous Beans, the robot used in the experimental runs, and shown in Figure 5.2.

The behavioural structure of the control system used for Dangerous Beans is depicted in Figure 5.3. The behavioural modules with a darker shade are dependent on those with a lighter

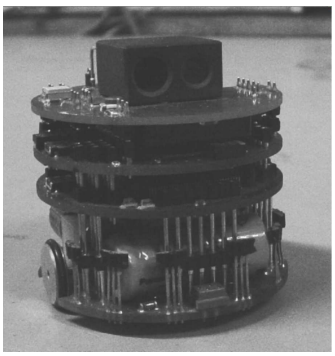


Figure 5.2: Dangerous Beans

shade, either because they rely on their output, or because they rely on the behaviour emergent from their execution. The dashed arrows represent corrective relationships (where higher level modules provide corrective information to lower level ones), and behaviours shown in dashed boxes are present in multiple instantiations. Solid arrows represent input–output relationships.

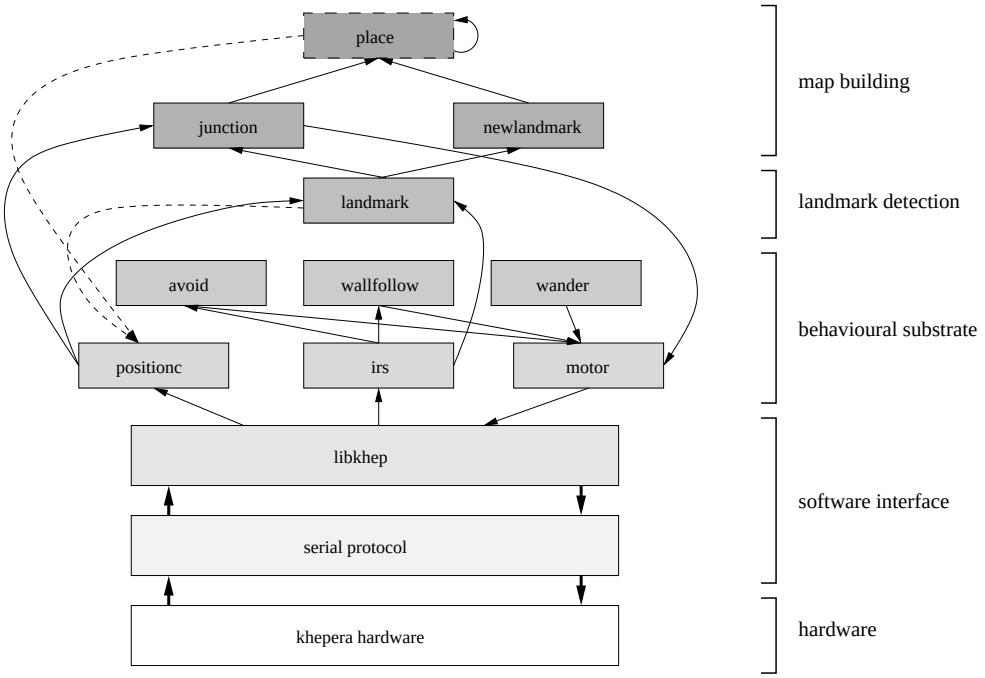


Figure 5.3: Dangerous Beans: Behavioural Structure (Map Building)

The hierarchy in Figure 5.3 is divided into the following levels: hardware, software interface, behavioural substrate, landmark detection, and map building. The following sections describe each of these layers in greater detail.

5.3.1 Hardware

Dangerous Beans is a standard issue K-Team khepera robot (serial number 97–401) with a pixel array extension turret. The khepera has a diameter of approximately 55mm, eight infra-red proximity and ambient light sensors, and two wheels with incremental encoders (K-Team SA, 1999b). The pixel array extension provides a single line of 64 grey-level intensity pixels with a viewing angle of 36^{o} (K-Team SA, 1999a). The infra-red sensors were used for obstacle avoidance and wall following, while the pixel array was used for puck detection. Figure 5.4 shows this sensory configuration, with the infra-red sensors numbered from 0 to 7, and indicates the angle of view of the pixel array turret.

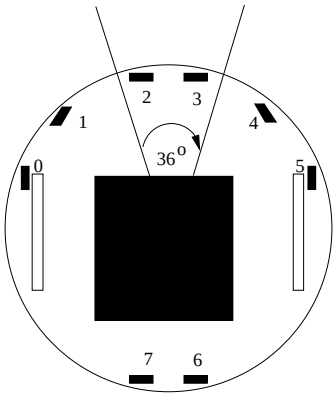


Figure 5.4: Dangerous Beans: Sensory Configuration

Although the khepera has an on-board processor, for the purposes of this research it was controlled through a serial cable at 38.4kbps, which supported a simple communication protocol allowing programs on a host computer to interface with it. This approach was taken because it made recording rich data about the robot's control state possible.

5.3.2 Software Interface

The software used to interface to Dangerous Beans was written in C, and run off a standard DICE Linux machine (`balliol`) in the Mobile Robot Lab, connected to Dangerous Beans with a serial cable suspended from a counterbalanced swivelling tether.

As a first layer of abstraction, a library of calls (`libkhep`) was developed that communicated with the serial port and covered all of the required khepera commands, and was made thread-

safe. The library was tested through the development of sensor monitoring software and was able to perform sufficiently well even while servicing several active threads in tight loops.

Each behaviour was allocated its own thread and all of the behaviours were run asynchronously, with no attempt at scheduling, which kept the behavioural processes parallel and loosely-coupled (Pfeifer and Scheier, 1999). Communication between threads where necessary was accomplished using global variables, with one thread writing to the variables and possibly multiple threads reading. These variables were typically not made thread safe as they were usually atomic, and since control was reactive and continuous the effect of any incorrect values encountered were transient. The only exceptions to this were sets of compound global variables required to be updated atomically, such as the global landmark flags. Finally, each behaviour had a wait flag which caused the behaviour to pause while it was non-zero, allowing behaviours to suppress each other.

5.3.3 Behavioural Substrate

The behavioural substrate developed for Dangerous Beans was required to produce sufficiently robust wall-following behaviour to allow for consistent and reliable landmark detection and map building. To this end, four behavioural layers were developed – one for handling infra-red sensing and averaging, motor output and position estimation, one for obstacle avoidance, one for wall-following, and one for map building. The first two can be considered a standard behaviour-based substrate and are described below.

5.3.3.1 Movement and Sensing

Two behaviours, `irs` and `motor`, were created to handle the interface between other behaviours and the robot's sensors and actuators.

The `irs` behaviour obtained data from the infra-red sensors and posted the average of the last three sensor readings to a global structure accessible by all threads. Averaging proved necessary because some of the sensors were fairly noisy, and because wall-following proved difficult without accurate readings at low activations.

The `motor` behaviour consulted a global structure which contained left and right motor fields for each behaviour that could issue motor commands, and sent the sum of these motor com-

mands to the khepera. Although the behaviours were mostly active under different conditions, and thus did not require much coordination, it proved best to simply add their motor outputs during transition phases in order to avoid brief episodes of erratic behaviour. The `motor` behaviour also checked each behaviour's `wait` flag and did not include motor output from waiting behaviours, thereby allowing for the immediate suppression of the effects of behaviours that had not yet checked their wait flags.

The `positionc` behaviour performed dead-reckoning position estimation based on encoder readings from the khepera's wheels. The simple tracking equations suggested by Lucas (2000) were used for dead-reckoning:

$$\theta_{t+1} = \frac{r_d - l_d}{d} + \theta_t$$
$$x_{t+1} = \frac{r_d + l_d}{2}\cos(\theta_{t+1}) + x_t$$
$$y_{t+1} = \frac{r_d + l_d}{2}\sin(\theta_{t+1}) + y_t$$

where r_d and l_d are right and left wheel displacements, respectively, and d is the diameter of the robot. Since the above equations for x and y are approximations based on the assumption that the angular change is small, the position estimation algorithm was run with a delay of $5\mu s$ between calculations, allowing for very small adjustments to be made at each point. The behaviour wrote the estimated position and landmark type detected at each position to a file which was used to produce dead-reckoning maps of the type shown in Figure 5.6.

5.3.3.2 Obstacle Avoidance

Two behaviours were used to implement obstacle avoidance: `wander` and `avoid`. The `wander` behaviour simply kept Dangerous Beans moving forward by setting its left and right motor outputs to a given global speed.

The `avoid` behaviour implemented safe-distance thresholding. Each of the forward sensors has a safe distance level associated with it, and the robot detected an obstacle on one of the sensors when its reading reached this level. A single, global threshold could not be used – the front and angular sensors require low (but different) thresholds because they would likely approach the wall relatively directly, and require immediate attention, whereas the lateral sensors require very high thresholds (and gentle turns) to avoid breaking away from a wall that the robot might be trying to follow.

Figure 5.5 shows the threshold values used for each of the front sensors. Note that a much lower

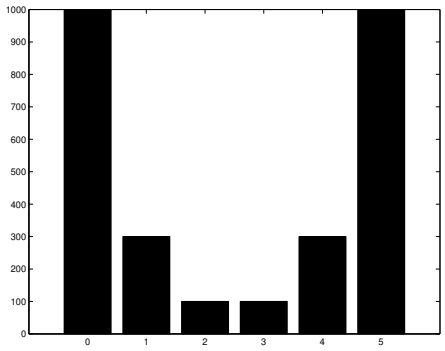


Figure 5.5: Front Sensor Obstacle Avoidance Thresholds

threshold could have been used for the lateral sensors had they had a wider range; however, because the khepera's sensors have approximately 25mm between a minimum reading of 0 and a maximum reading of 1023 on the kinds of surfaces used in the arena (K-Team SA, 1999b), an extreme value had to be used to minimise the chances the robot would veer away from a wall due to a very small movement or sensor noise.

When a single one of the forward sensor thresholds was exceeded, the robot turned on the spot away from the incoming obstacle. If *both* of the forward sensor thresholds were exceeded, then the lateral sensors were checked for any activation at all, and if it was present (indicating an obstacle on one side of the robot) the robot turned in the other direction. This achieved good cornering behaviour, and avoided the robot turning into walls that it had been following. If neither lateral sensor was active then the robot turned left by default.

When either of the angular sensor thresholds were exceeded, the robot briefly backed up and simultaneously turned gently away from the active sensor. This allowed the robot to turn slightly away from a wall without potentially losing it, at the cost of slightly jerky motion during corrections. Experiments where angular sensor activation caused the robot to simply turn showed that doing so would in many cases lose a wall because a delayed drop in sensor reading as the wall receded would cause it to turn for too long.

When either of the lateral sensor thresholds were exceeded, the robot turned gently away from the active sensor, tracing out a slight curve that did not move the robot away from the wall too quickly for the wall-following behaviours to recover. The resulting emergent behaviour in fact in most cases achieved wall following without any explicit attempt at doing so; however, because there was no attempt to keep the wall at a constant distance the robot was prone to

veering slightly away from the wall due to sensor noise and wandering off.

5.3.3.3 Wall Following

Wall-following was implemented using a single behaviour, `wallfollow`, which became active whenever either lateral sensor read above 100 and below 1000, but was suppressed when either of the forward sensors were active, in order to avoid interfering with cornering behaviour.

The behaviour attempted to keep the sensor reading at a target value of 550. The problem of keeping the sensor at such a value can be considered a steady state problem, and thus a standard Proportional-Integral-Differential (PID) control implementation was an obvious candidate. However, implementing a full PID controller turned out to be infeasible, because of the noise characteristics and short range of the khepera's infra-red sensors, and the fact that the khepera's wheels where under integer control at a fairly low speed.

Extensive testing showed that a simpler, modified approach was able to best handle the characteristics of the khepera and the task requirements. The approach taken used a proportional attraction force and a small constant repulsion force. The constant repulsion force ensured that the khepera did not lose the wall by turning away from it too quickly, while the proportional attraction force acted to quickly correct the robot's movement when it was moving away from a wall. The probability that the robot would not respond strongly enough to an encroaching wall and thus collide with it was made negligible by the near-wall-following behaviour of the obstacle avoidance substrate and its use of a lateral threshold.

Control forces were calculated for the active lateral sensors, summed, and converted to a proportion of the maximum possible repulsive force. The result was used to modify the direction of the robot – half of this proportion of the target speed of the khepera was added to one wheel and half subtracted from the other, moving it in a shallow curve towards the target value, or towards an equilibrium in the case of walls on either side.

5.3.4 Landmark Detection

The `landmark` behaviour performed landmark detection, and broadcast the current landmark type, heading, and length (number of consecutive readings). The landmark type took on values of either right wall, left wall, corridor, or empty space, and the current heading was given as

one of 0, $\frac{\pi}{2}$, π or $\frac{3\pi}{2}$ radians. The behaviour made use of the dead-reckoning position computed by `positionc` to estimate the angle of a wall, and then supplied a corrected angle back to the position estimator in order to minimise dead-reckoning angular error in the absence of a compass.

The `landmark` behaviour used a simple statistical approach similar to that given in Matarić (1990). A set of 50 thresholded samples were taken from the left and right lateral sensor, with a $10\mu s$ delay between samples. Each sample was thresholded at the lower bound for wall-following activation (an infra-red sensor reading of 100). The landmark was determined to be a corridor if at least 25 samples showed left activation and 25 showed right activation; failing that, it was determined to be either a left or right wall if at least 30 samples of the relevant side were above the threshold. If neither condition was met the landmark type was set to the default value of free space.

A new landmark was detected if the type of landmark changed, or if the estimated angle of the robot differed from that of the currently active landmark by 0.8 radians, which is half the distance between the expected wall angles. If the landmark did not change, then the landmark length count was increased. The estimated angle of the landmark was selected as one of the four orthogonal directions that walls are expected to lie along, with the selection performed using either the current angle (if the landmark length count was less than 4) or the average of all measurements so far, not including the first and last set of measurements. The use of averaging only past a certain length and the practice of ignoring the first and last estimates was required because the angular estimations obtained early were typically significantly off because of the approach angle of the robot and the possibility that it had just followed a curved corner and was not yet moving straight.

Landmarks of length 4 or more were used for angular correction, whereby the difference between the expected angle of the landmark and the average angle obtained was used to correct the current angle estimation. This was required because the robot required accurate angular estimation over time and dead-reckoning without correction could not maintain this. The outputs of two position processes, one without landmark-based correction, and one with landmark-based correction, are shown in Figure 5.6. The colour of the lines fade over time, indicating the robot's movements as time progresses, and the dots in the second graph indicate the robot's position when it made adjustments. Adjustments where the difference between the estimated and observed angle was less than 0.05 radians were ignored.

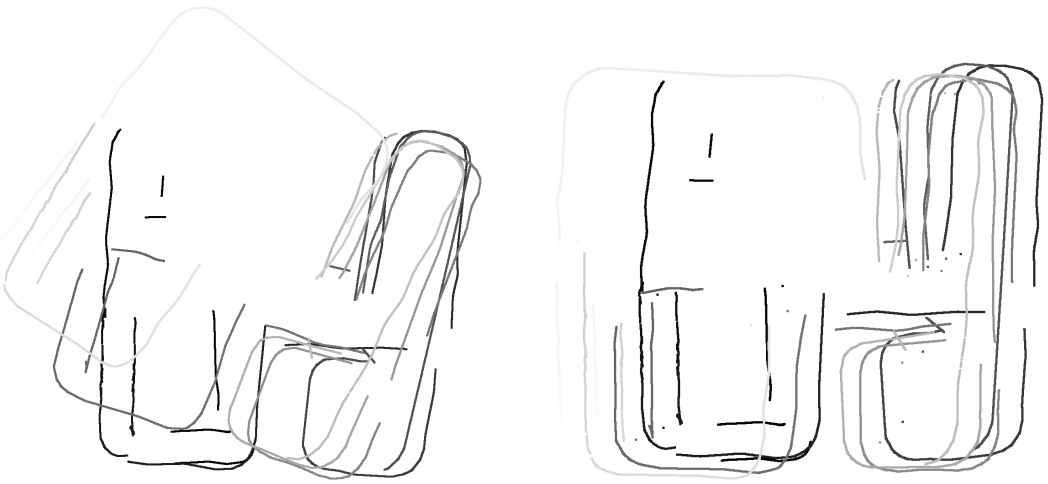


Figure 5.6: Uncorrected and Angle-Corrected Dead-Reckoning Maps

Note that in both cases there is definite horizontal and vertical dead-reckoning error. However, in the case of position estimation with correction, the angular displacement does not degrade consistently over time. An examination of the upper right corner trace indicates that although the position estimates differ over time, the traced lines stay nearly parallel, whereas in the figure on the left there is a definite angular degradation over time. The accuracy achieved using the corrected angular estimates is sufficient for landmark discrimination in this case, where walls are known to be horizontal and vertical only.

5.3.5 Map Building

The layer of behaviours responsible for map-building were required to maintain a distributed map by creating new "place" behaviours for novel landmarks and linking places together when they appeared sequentially. In addition, because the choices the robot was allowed to make were less restrictive than those allowed in Matarić (1990), and could lead to it encountering walls halfway through, nodes which were found to represent different parts of the same landmark were required to be merged.

Therefore, each place was allocated its own `place` behaviour, along with its own thread. Each `place` behaviour maintained a "landmark descriptor", which consisted of the type, angle, estimated coordinates, and connectivity information of the corresponding landmark. The descriptor was used by each `place` behaviour to continuously compute the probability that it

corresponded to the current landmark. The `place` behaviour with the highest probability was judged to be the winner, although few had non-zero probabilities at the same time.

Each `place` behaviour maintained a linked list of transitions, which stored the `place` behaviours that became active immediately after them, the type of turn (left, right, or straight, with an extra direction modifier to indicate which end of the landmark the turn was from) that resulted in the transition, and how many times that combination had occurred so far. In a similar manner, each `place` behaviour also kept a list of the other behaviours that had directly preceded it. When one behaviour became active, the `place` behaviours that had at some point followed it were marked as "expecting".

Each `place` determined the probability of it being the current landmark using a combination of landmark type, angle and dead-reckoning distance. Place behaviours not of the correct type and angle immediately set their probabilities to zero. Those with the correct type and angle calculated a distance metric using the following equation:

$$1 - \frac{x_d + y_d}{200} \tag{5.1}$$

where x_d and y_d were the shortest x and y distances from the landmark, respectively. Probabilities of less than zero were rounded up to zero. The distance metric gives probabilities inversely proportionate to distance, reaching zero at about 20cm away.

Although the model given in Matarić (1990) uses expectation as a deadlock breaker before dead-reckoning (which was seldom necessary), because of the higher branching factors and more complex maps created here, dead reckoning was required fairly frequently. Therefore, expectation was not used here; however, it could easily be used to modify Equation 5.1 to give expecting nodes higher probabilities.

The `newlandmark` behaviour was responsible for detecting when no place behaviour had a sufficiently high probability of corresponding to the current landmark and allocating a new one for it. For simplicity, the `newlandmark` behaviour also determined which place behaviour was the current best, and when to merge landmarks.

When two landmarks are strongly active at the same time, indicating that they could both potentially account for the same landmark, `newlandmark` checked to see whether or not they were of compatible types and angles, and whether or not their long axes overlapped. If they did, then they were likely to be duplicate landmarks and were merged. Merging involved calculating

new bounds for the landmark and merging the transition lists associated with the two places. Duplicate landmarks were artifacts of the fact that Dangerous Beans sometimes encountered a wall half way through, and therefore only created a landmark behaviour covering half of it, allowing for a new behaviour to erroneously be created if the wall was later encountered on the unexplored side. This problem does not occur in the model used by Matarić (1990) because of its more strict wall-following behaviour, but it is a significant problem here. Fortunately the merging procedure adopted here solved it in all observed cases.

Although `newlandmark` performs a somewhat centralised role, (and corresponds to a "horizontal" behavioural module coordinating several "vertical" behavioural modules, according to Bryson (2002)) these roles could be implemented in a distributed system using simple suppression and excitation relationships. However, this is relatively difficult in a threaded model, and thus was not attempted here.

In addition, each `place` behaviour was responsible for correcting the current position of the robot. When a landmark was first encountered, the `place` behaviour created for it recorded the average dead-reckoning coordinate along its long axis and stored it. Later, when the same landmark was traversed, a similar average was kept, except that when the place behaviour was no longer active, it corrected the position of the robot according to the difference between the two averages. Figure 5.7 shows two graphs from the same run, where the graph on the left was produced with just angular correction, and the one on the right was produced with landmark-based position correction.

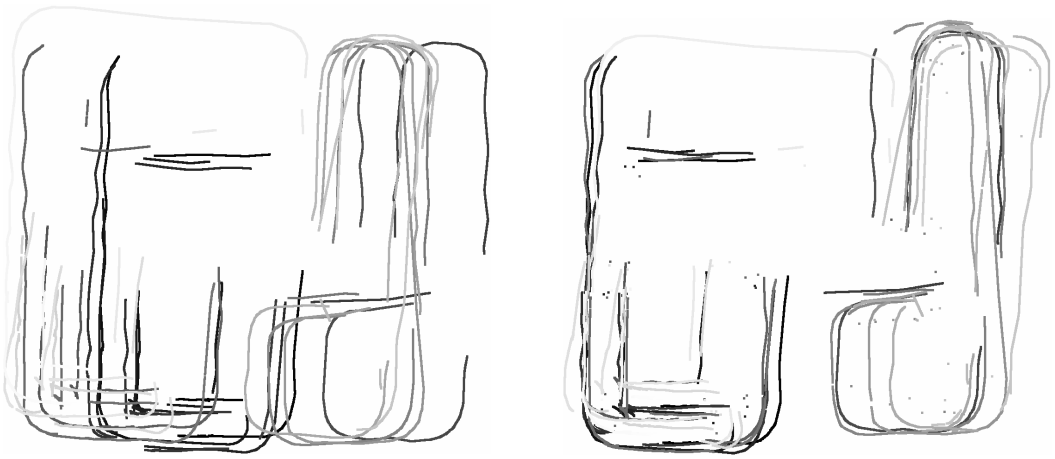


Figure 5.7: Angle-Corrected and Fully-Corrected Dead-Reckoning Maps

It is clear from Figure 5.7 that without landmark-based correction, dead-reckoning error would cause the robot's perceived position to drift. This would, in turn, cause mapping errors because it would defeat the distance measure used to disambiguate between landmarks. This simple approach proved sufficient for the simplified environment used in the experiments; however, in a more general environment, a more complex approach would be required (e.g., Choset and Nagatani (2001)).

Figure 5.8 is a visualisation of the distributed mapping data produced by Dangerous Beans on the first test arena. Landmarks are represented by rectangles, each with a central circle and two end circles. Corridors have two rectangles with the circles placed between them. Lines represent transitions, where a line from the circle at the end of one landmark to the circle in the middle of another indicates that Dangerous Beans has been able to move from the first landmark to the second. The map contains 17 landmarks and 32 edges, although some edges are not distinguishable here because they have different turn types but are between the same landmarks.

The slightly exaggerated length of all of the landmarks is an artifact of the landmark recognition algorithm used. This means that some landmarks may appear to overlap (for example in the bottom left corner) but are actually just close together.

Note that the distributed map has more information than is shown in Figure 5.8; in particular, it has direction data (which is indicated to some extent in Figure 5.8 by the side an outgoing line leaves a landmark from) and frequency data. The map visualised in Figure 5.8 therefore contains all of the information required for the addition of a distributed reinforcement learning model.

5.4 Summary

This chapter has outlined the properties of the physical arena constructed for the experiment outlined in Chapter 4 and the robot designed to perform distributed map building in it. The behavioural substrate, map building and navigation mechanisms of the robot were described, and shown to be sufficiently robust to provide the basis for a distributed reinforcement learning layer. The following chapter describes the development of such a layer, and presents the results of its use on the three experimental arenas described in Chapter 4.

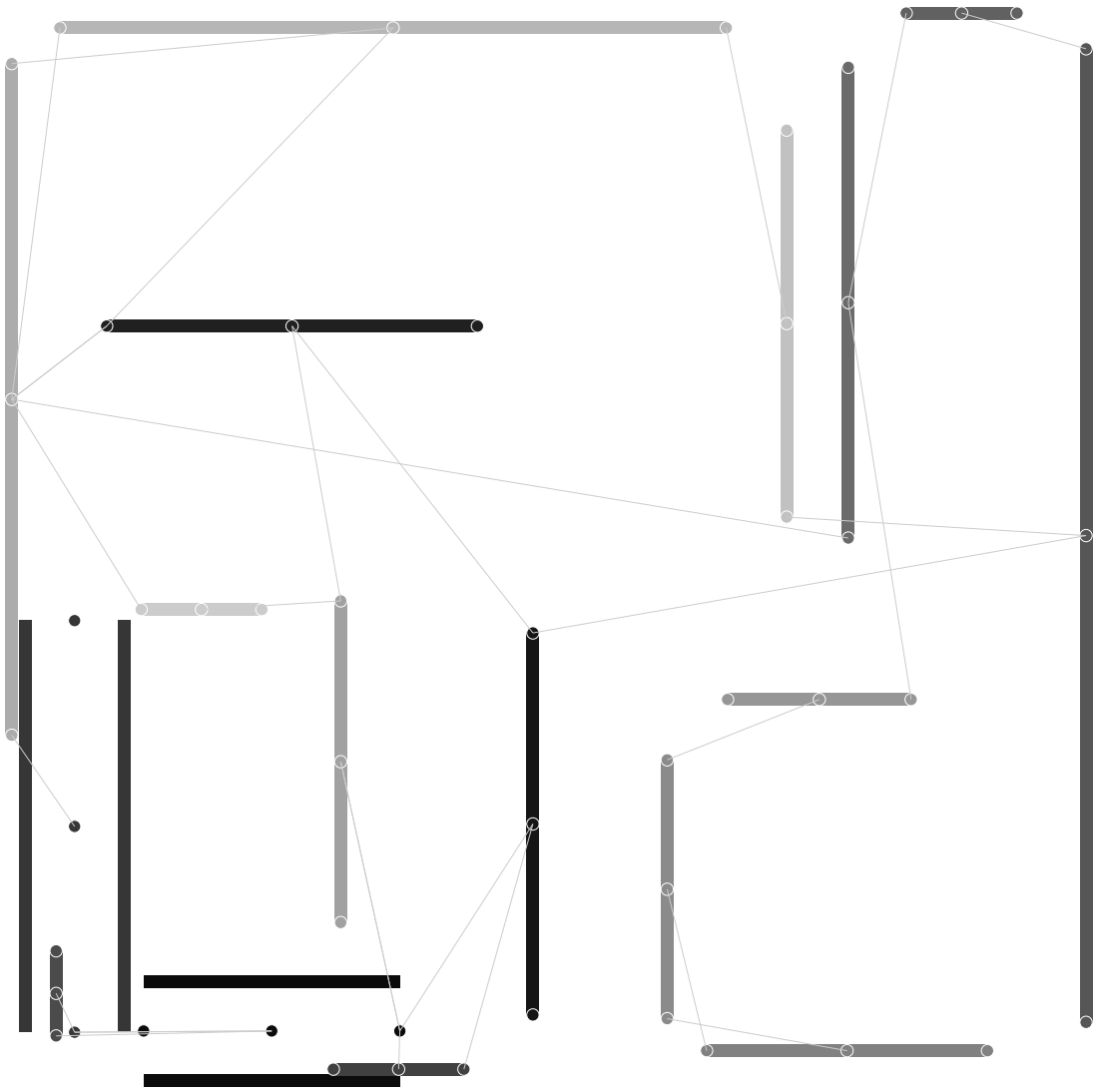


Figure 5.8: A Distributed Topological Map of the First Arena

Chapter 6

Distributed Reinforcement Learning in an Artificial Arena

6.1 Introduction

The critical test for a learning model that claims to be able to improve the performance of an existing robot system is whether or not it can perform as required in the real world, in real time. This chapter describes an implementation of one variation of the reinforcement learning model described in Chapter 3, and presents the results of its use in the experiment described in Chapter 4. The results show that the distributed, embedded and asynchronous reinforcement learning model performs significantly better than both a random choice algorithm and a standard (synchronous) reinforcement learning algorithm, and that it is capable of generating complex, adaptive behaviour in a real robot in real time.

The following section describes the additional functionality added to the distributed map learning robot presented in Chapter 5 in order to implement the reinforcement learning model. The section after that describes and analyses the results of the experiment, and the final section draws conclusions from these.

6.2 Implementation

Chapter 5 described the development of Dangerous Beans, a robot capable of distributed map building in an artificial arena. This section describes the additional control structures added to Dangerous Beans to enable it to perform distributed reinforcement learning over its distributed topological map, and to record the data required to provide the results presented in the remainder of this chapter. The behavioural structure used in the experiments is shown in Figure 6.1, with the behaviours added to the structure given in Chapter 5 shown with darkly shaded boxes.

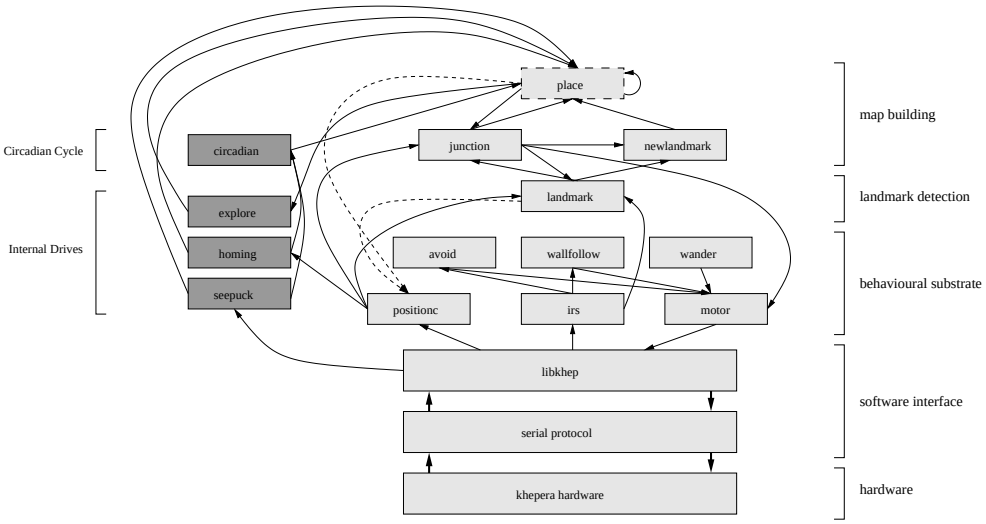


Figure 6.1: Dangerous Beans: Behavioural Structure (Reinforcement Learning)

Although for the most part these changes were simply added on top of the control structures already present in the robot, many of them required small behavioural or architectural changes to the existing software. The final part of this section will describe the significant changes made, and give the reasons for each of them.

6.2.1 Obtaining Reward

In order to express the three drives required in the experiment, three reward behaviours were added to Dangerous Beans. Each behaviour exposed a global variable that could be read by other behaviours in order to obtain reward information. All of the behaviours updated their rewards using the equations given in Chapter 4, with a one second delay between updates.

The `seepuck` behaviour was responsible for determining when the robot was facing a puck, and should receive a puck reward. Each puck was covered in black tape at the level of the pixel array, and showed up as a dark band with a light background in the pixel array. The behaviour therefore used a simple averaging thresholding algorithm (Ballard and Brown, 1982), where pixels that were at least a threshold of 80 (out of 256) below the average intensity of the image were marked as dark. A dark band was considered a puck when it had no light interior pixels, a minimum size of 7 and maximum size of 20 pixels, and was not on the edge of the image. The final condition was required because the images produced by the pixel array often had dark edges.

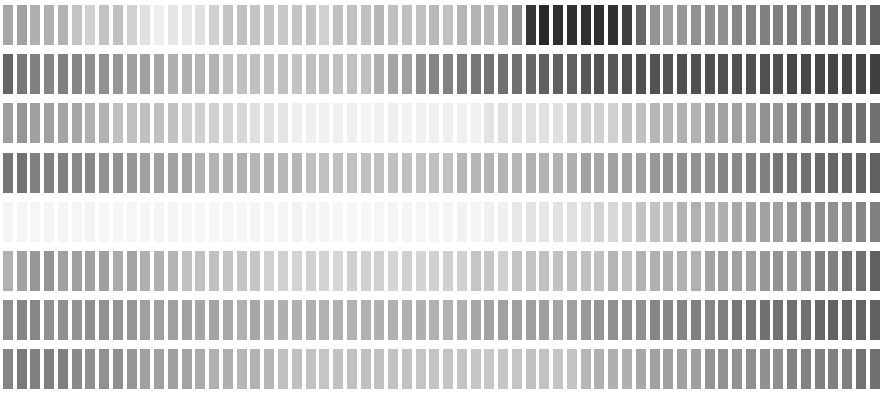


Figure 6.2: Sample Pixel Array Images

Figure 6.2 shows some sample images from the pixel array while Dangerous Beans was in the arena. The top sample was taken when the robot was facing the puck, and the rest were taken at arbitrary positions in the arena. Although there are significant dark areas in the other samples (especially near the edges), only the sample with the puck in sight has the distinctive black band against a light background.

The `seepuck` behaviour also inhibited puck reward for approximately 20 seconds after a puck was detected, in order to avoid multiple rewards being issued for the same puck sighting.

The `homing` behaviour checked whether or not the robot's position (as estimated by `positionc`) was within some threshold in both directions of the robot's original location, so that any location within this boundary was considered home. The thresholds were set separately for each arena in order to correspond to its home area. The behaviour required the robot to be at least 10cm outside the area and return before allocating reward again.

Finally, the `explore` behaviour was given the number of times each transition had already been taken as it was taken again, and using this along with the overall average computed from the set of `place` behaviours, determined a transition reward according to the exploration reward equation given in Chapter 4.

6.2.2 Circadian Events

The `circadian` behaviour was responsible for keeping track of the current cycle and active phase of the robot. It monitored the reward behaviours introduced in the previous section along with a timer, and switched the robot's state between the foraging, exploring, and homing modes when necessary. The behaviour exposed a global variable representing the current phase (and thereby active desire) for other behaviours to use when making decisions.

6.2.3 Making Choices

The `junction` behaviour was extended to allow place behaviours to signal decisions they wanted made (in the form of requested turns) by allowing them to post their requests to a global variable that was checked every time a decision had to be made. The global variable was timestamped in order to avoid old decisions being taken after they were no longer valid. The `place` behaviour was modified so that it only posted once per activation (unless the current drive changed while it was active, in which case it was allowed to post again), and according to whichever control strategy (random or one of the reinforcement learning models) was being used at the time.

The `junction` behaviour executed the requested turn if possible; some turns had to be ignored when they could not be executed because of the presence of an adjoining obstacle. Therefore, each turn at each place had an associated counter, which was incremented when the turn was successfully taken and decremented when it could not be. When this counter reached -2 the turn was banned, and not considered for any further decision making or reinforcement learning purposes. This was required because occasionally attempted illegal turns were made which caused the robot to "bounce" off the wall and find another wall, resulting in an apparently legal transition.

The `junction` behaviour was also responsible for determining when the robot was headed along the wall in the wrong direction given the decision made, and reversing the robot's di-

rection without losing contact with the wall. A simple turn waiting for the activation of the currently inactive lateral sensor proved sufficient for this, although the turn could only be made reliably once the same landmark had been broadcast three successive times by the landmark detector. If attempted earlier, the turn would often lose the wall because Dangerous Beans' wall-following behaviour was still stabilising.

6.2.4 Place and Transition Values

Since the robot was likely to spot a puck shortly after making a transition, and could not guarantee that simply by being at a particular landmark it would see the puck, reward was allocated to transitions rather than places. Each transition received the reward obtained from the time that the robot left the landmark it was from, to the time that the robot left the landmark it was to.

As explained in Chapter 5, each `place` behaviour maintained a list of transitions originating from it, containing turn, frequency and place data. In order to record the reward obtained by each transition, each `place` behaviour kept a record of the relevant reward values as soon as it became inactive. The transition made was then noted, and when the place that it led to became inactive again, the transition received the difference between the initially noted reward values and the reward values after the end of the place it had led to. Each transition kept a total of the reward it had received along with the total number of times it had been taken, and the number of those times where a negative reward was received.

The update equation used for the asynchronous reinforcement learning model was run by each place behaviour at all times for all turns, and was the same as the example ATD update equation given in Chapter 3, with $\gamma = 1$:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha(r_{s,a} + E_{s,a}\{V(s_{t+1})\} - Q_t(s,a))$$

where

α	is the learning step parameter, set to 0.1[1].
$Q_t(s,a)$	is the value of taking action (turn) a at state (place) s at time t.
$r_{s,a}$	is the expected reward received for taking action a at state s.
$E_{s,a}\{V(s_{t+1})\}$	is the expected state value after action a at state s, at time t.

[1]This is the most common value used in Sutton and Barto (1998). Due to time constraints, no systematic evaluation of its effect was performed.

Each `place` stored the Q values for each of its possible turns, and during the update $E_{s,a}\{V(s_{t+1})\}$ was calculated for each turn by computing the weighted sum of the values of each state encountered after taking turn a at state s. The expected reward term $r_{s,a}$ was computed for each action as the average reward obtained over all executions of the transitions using turn a from the state. For the exploration reward function, the estimated reward was computed directly from the equations given in Chapter 4, since previous exploration rewards for a particular turn were not useful in estimating its current value.

Since the task was effectively episodic, when a transition had a positive reward its contribution to the expected value of the following state was not included. This has the same effect as considering positive rewards to end the episode, and prevented positive feedback loops where states could have obtained infinite expected values.

In the synchronous update case, the value function for each state-action pair was only updated immediately after a transition from the state using the action was completed, and instead of an average reward, the reward obtained was used directly. The update equation used for the synchronous case was:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha(r_{t+1} + V(s_{t+1}) - Q_t(s,a))$$

where

α	is the learning step parameter, again set to 0.1.
$Q_t(s,a)$	is the value of taking action (turn) a at state (place) s at time t
r_t	is the expected reward received at time t.
$V(s_t)$	is the value of the state active at time t.

Since the value of each state was taken as the expected value of the maximum action that can be taken there, the synchronous case is equivalent to Q-learning (Watkins and Dayan, 1992).

In order to encourage exploration, actions that had not yet been taken from a given state were assigned initial values of 50 for both homing and puck rewards. Initial exploration rewards were set to 200, as required by the exploration reward function given in Chapter 4. All initial reward estimates were immediately replaced by the received reward for the asynchronous model.

For the reinforcement learning models, when a `place` behaviour became active, it would post a decision to the `junction` behaviour using the action with the highest action value, with ties broken randomly. When all of the action values available were negative, or when the requested

action could not be taken, a random action was chosen. In all cases, only legal turns (those allowed by landmark type and so far not found to be impossible) were considered. In the random decision case, one of the legal turns was simply picked at random and posted to the `junction` behaviour.

6.2.5 Data Capture

The graphs and analysis presented later in this chapter required data from several different processes running on Dangerous Beans. The following information was captured during run time and written to disk:

- Robot position (as estimated by `positionc`).
- Circadian events (from the `circadian` behaviour).
- Reward levels (from the `seepuck`, `homing` and `explore` behaviours).
- Place descriptor and activation data (from each `place` behaviour).
- Transition data (from each individual transition).
- Average reinforcement learning value update sizes (from each transition).

All of the data was written to file along with a timestamp by the processes responsible for generating it. A set of scripts was written in `awk` and `bash` to convert this data to Matlab program code so that it could be examined and manipulated, and these scripts were parametrised so that a time interval could be specified, allowing for the extraction of subsets of the data.

6.2.6 Modifications to the Original Map-Building System

The original distributed mapping system required modification in order to successfully perform the experiments presented in Chapter 4. Three primary difficulties were encountered.

First, the addition of the reinforcement learning, decision making, and reward modules required architectural changes to the distributed mapping software. These changes affected the internal structure of the control system but did not substantially affect the robot's behaviour.

Second, the added ability of the robot to turn when it was headed in the wrong direction disturbed the functionality of some of the already existing modules. If the robot was following a

wall on its right and then turned, because of the slightly delayed nature of the landmark detector, a brief "ghost wall" or corridor would be detected by the landmark detector on the right of the robot after it had turned, when in fact the wall was now on the left. Furthermore, after reversing direction the number of consecutive landmarks detected was no longer an accurate indicator of the length of a wall or corridor, because it would have been reset half way through. These problems arose not because the added behaviour explicitly disturbed any of the already existing control software, but because some parts of the existing software relied on the emergent behaviour of the control system in order to function correctly. This introduced subtle and difficult to isolate errors into the control system which were only discovered through extensive testing, and were solved through the suppression of the disturbed behaviours during the execution of a reverse turn.

Finally, the use of the random decision making agent introduced significant noise to the dead-reckoning system. This occurred because the robot would often repeatedly double back near corners, thereby avoiding angular and landmark-based correction because it was not at any single landmark for long enough for either to take effect. Because of this, and because of the loss of usefulness of consecutive landmark detections, both the landmark-based and angular correction schemes were modified to supply correction values during contact with a landmark of sufficient length, rather than after. In addition, because of "landmark slip" (where place position estimates became wider over time because of dead-reckoning error) landmarks considered for merging were required to overlap by at least 7cm. The new correction scheme provided equivalent results to the old one in the reinforcement case, and superior results in the random case. However, because of the inherent difficulty of accurate position estimation, the correction mechanisms failed in some cases, and runs where this occurred were restarted. In most cases failures occurred because of inaccurate angle estimates over long empty spaces where the robot could not obtain corrective information, and were primarily the result of angular errors. This could be solved with the addition of a direction sense (e.g., a polarisation compass (Schmolke and Mallot, 2002)), or the use of a method for landmark disambiguation not based on dead reckoning (e.g., neighbourhood characteristics (Dudek *et al.*, 1993)).

6.3 Results

This section presents the results of using the robot control strategies developed in this chapter in the three experimental arenas described in Chapter 4. The following three sections describe the results obtained for each arena individually. Each time, the average puck and homing rewards over time for each model are compared and discussed, along with a qualitative evaluation of the relevant aspects of the robot's behaviour, and some examples of typical routes taken. The exploration component of each robot type's behaviour is also considered where relevant. The results for the three arenas are followed by a brief investigation into the convergence of the asynchronous reinforcement learning model, and a summary of the primary results.

6.3.1 The First Arena

In the first arena, both reinforcement learning models were able to learn to find direct routes to the single puck relatively quickly. Figure 6.3 shows the puck reward obtained over time (averaged over seven runs) for each of the models, with the error bars indicating standard error.

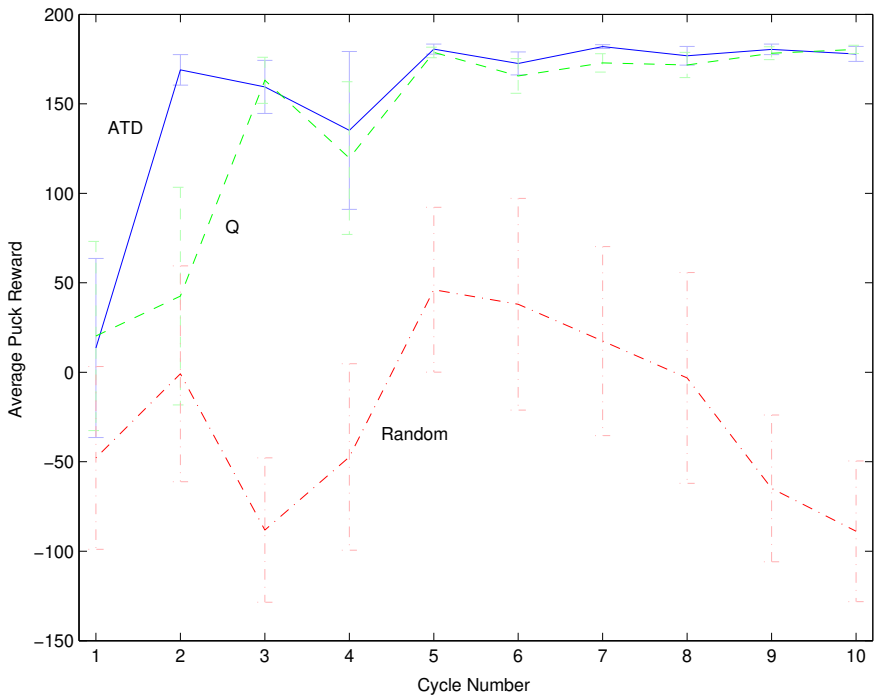


Figure 6.3: Average Puck Reward over Time: The First Arena

Since the path to the puck from the home area was relatively short, as expected both reinforcement learning models found and learned good solutions quickly, progressing from near-random results with a wide spread of reward values (indicated by the large error bars) in the first cycle to nearly uniformly good results (indicated by the very small error bars) by the fifth cycle. In contrast, the random control strategy performed poorly, resulting a low average reward with large error bars throughout.

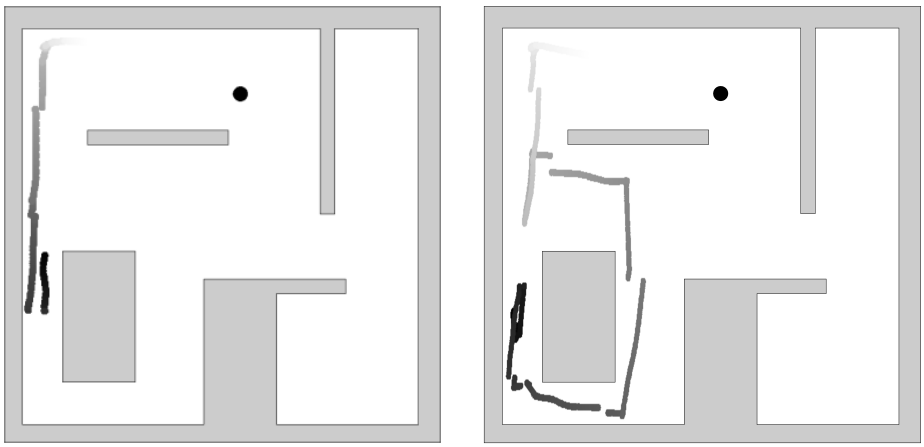


Figure 6.4: Learned and Random Routes to the Puck in the First Arena

The left part of Figure 6.4 shows the route learned in nearly all cases by both reinforcement learning models to the puck[2]. Note that the model has to turn around first after heading into the homing area, thus creating the double line in the lower left part of the path. The breaks in the path were caused by landmark-based correction. On the right is a sample path taken by the random algorithm to find the puck. The random algorithm does not move directly towards the puck and instead finds it by chance. Although this path is nevertheless quite short, this is because when the random agent wandered into the trap on the right or doubled back on itself repeatedly it virtually never encountered the puck before the end of the cycle.

Both reinforcement learning models were also able to return to the home area quicker than the random model, as indicated by Figure 6.5. Again, both reinforcement learning models found good solutions quickly and consistently. The asynchronous model appears to have been able to return to the homing area quickly even at the end of the first cycle, which is likely a result of an active exploration strategy and its ability to learn rapidly. The random model performed

[2]These figures and the similar ones that follow were obtained by the superimposition of dead reckoning position estimation and a scale drawing of each map. They should be considered reasonable path approximations only.

poorly, as is to be expected given the trap on the right of the arena.

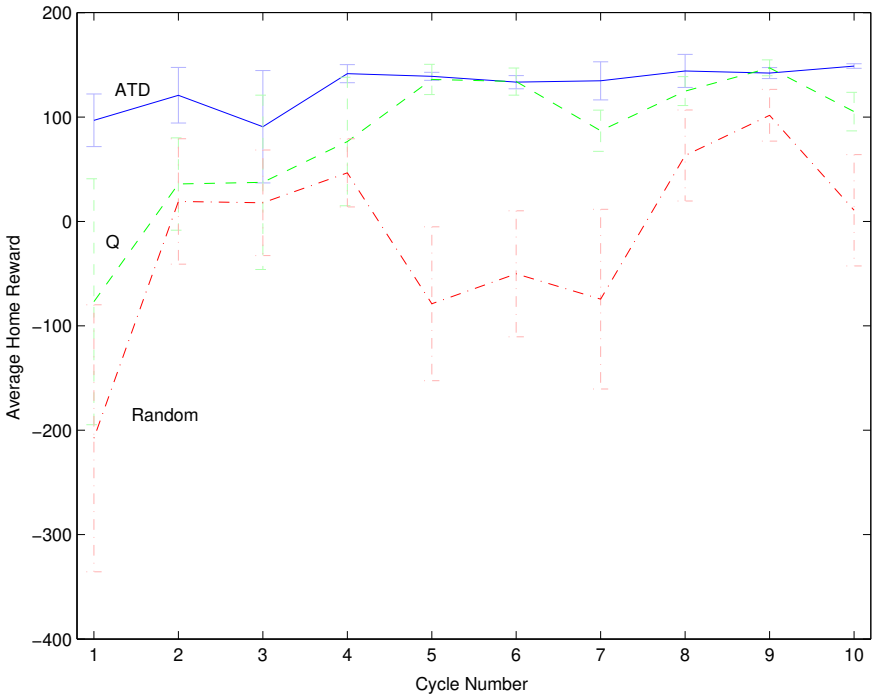


Figure 6.5: Average Home Reward over Time: The First Arena

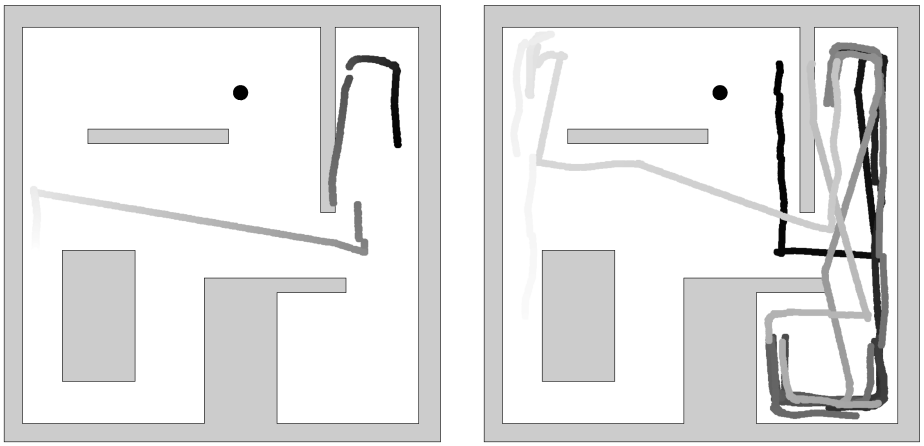


Figure 6.6: Learned and Random Routes Home in the First Arena

Figure 6.6 shows typical routes home for the reinforcement models (on the left) and the random agent (on the right). Note that the random agent gets stuck in the trap on the right for some time,

eventually wandering home, whereas the reinforcement learning agents escape immediately.

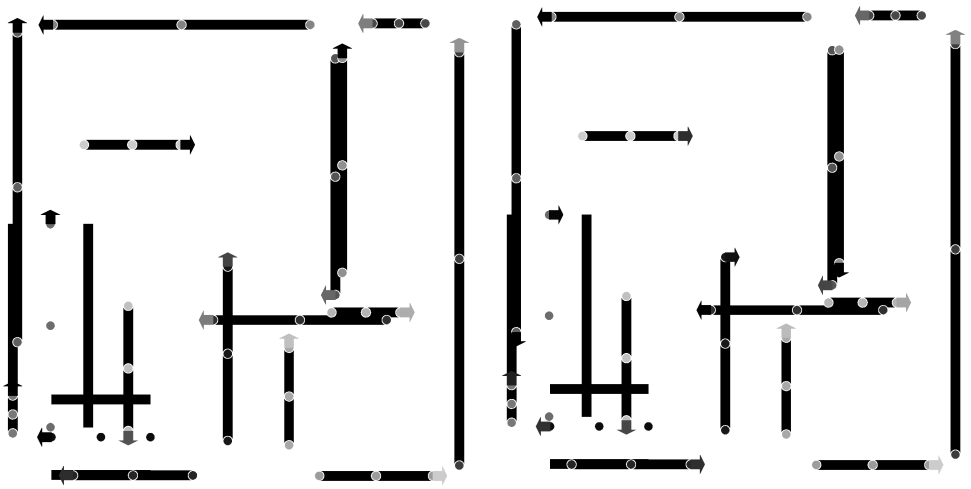


Figure 6.7: Preferred Transitions Maps for the First Arena

Figure 6.7 shows the robot's preferred transitions for the puck and homing phase at the end of one of the asynchronous runs, with darker arrows indicating higher values. It is clear from both of these maps that the reinforcement value complex has converged over the entire map.

Both reinforcement learning models appeared to explore effectively. Occasionally, however, an implementation problem caused the exploration behaviour of both reinforcement learning models to exhibit odd behaviour. The ledge at the bottom right of the arena was just long enough to be included in the distributed map but not long enough to allow the robot to turn around after moving to it. Because of this, the transition from the ledge to the vertical wall on its left could never be taken. This transition therefore obtained a very high exploration value, which caused the robot to repeatedly attempt to execute it. This sometimes resulted in the robot circling in the bottom right corner of the arena until the end of the cycle was reached, whereupon it went back to the home area. Although this problem affected the asynchronous model more than the synchronous one (presumably because the exploration value propagated through the map more quickly) it did not appear to negatively impact on the effectiveness of either model.

6.3.2 The Second Arena

As required, one of the pucks was removed from the second arena at the end of the fifth cycle in every trial. For both reinforcement learning models, the puck near the top of the arena was visited last in all seven runs and therefore removed, and for the random runs the same puck was removed in order to make the results maximally comparable.

As can be seen from the graph of the average puck reward obtained over time for the models in Figure 6.8, both reinforcement learning models learned to find a puck relatively quickly at first. However, when the first choice puck was removed at the end of the fifth cycle, both reinforcement learning models experienced a sharp drop in performance, along with a high variation in reward as indicated by the large error bars. Although the asynchronous model is able to recover and return to a consistently good solution by the ninth cycle, the synchronous model does not on average perform much better or more consistently than the random model by the end of the run.

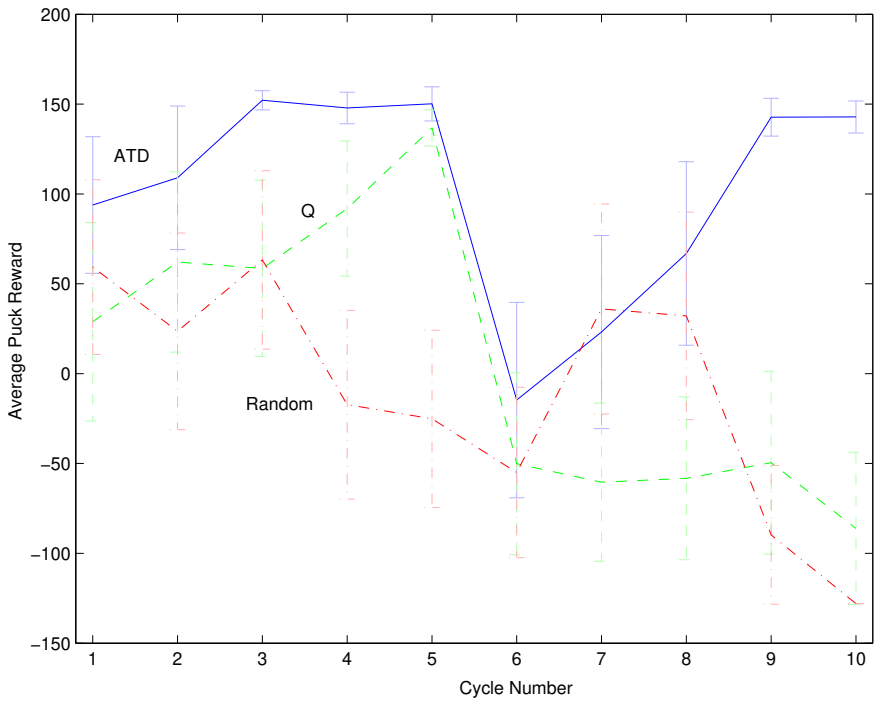


Figure 6.8: Average Puck Reward over Time: The Second Arena

The asynchronous model is therefore able to able to learn to adjust its value complex relatively

quickly in the face of a modified environment. It does this despite the fact that the expected values it calculates are averages over all rewards received, so that some residual puck reward must remain at any transition where a puck has ever been sighted. This could be remedied by the use of an average over the last few sightings or some similar measure, but it does not appear to have affected its performance here.

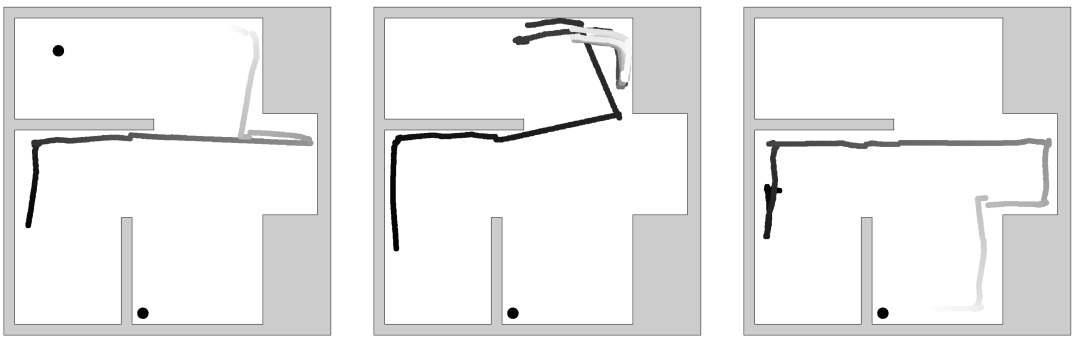


Figure 6.9: Learned Puck Finding Behaviour in the Second Arena

Figure 6.9 shows the puck finding behaviour displayed by the reinforcement learning models. The figure on the left shows an example of the puck finding path initially learned by both reinforcement models, and the figure in the middle displays the behaviour exhibited initially by both models after that puck has been taken away, where both robots repeatedly execute the transition that had previously led to a puck sighting. However, the asynchronous model is later consistently able to learn to take the alternate puck finding route, shown in the figure on the right, while the synchronous model is not.

Figure 6.10 shows the preferred puck transition maps after the eighth cycle for the asynchronous and synchronous models. The map obtained from the asynchronous model (on the left) has adjusted well to the removal of the top puck and now directs movement to the lower puck from everywhere in the graph (note that two pairs of walls in the left map appear to be on the wrong side of each other due to dead reckoning error). The map obtained from the synchronous model, however, has not adjusted as well and still contains regions where movement would be directed toward the transition where the puck has been removed.

Figure 6.11 shows that both reinforcement learning models were able to learn to get back to the home area relatively quickly. However, the synchronous learning algorithm experiences a drop in performance and increase in standard error from the sixth cycle, only recovering around the ninth cycle, while the asynchronous algorithm does not. This may indicate that

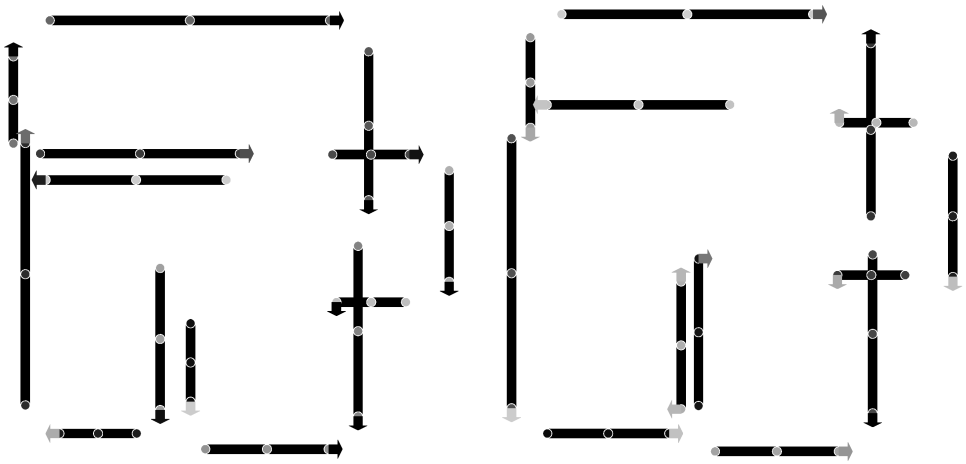


Figure 6.10: Preferred Puck Transitions Maps for the Second Arena after the Eighth Cycle

the synchronous algorithm is less robust than the asynchronous one, although this cannot be conclusively determined with the small number of runs performed.

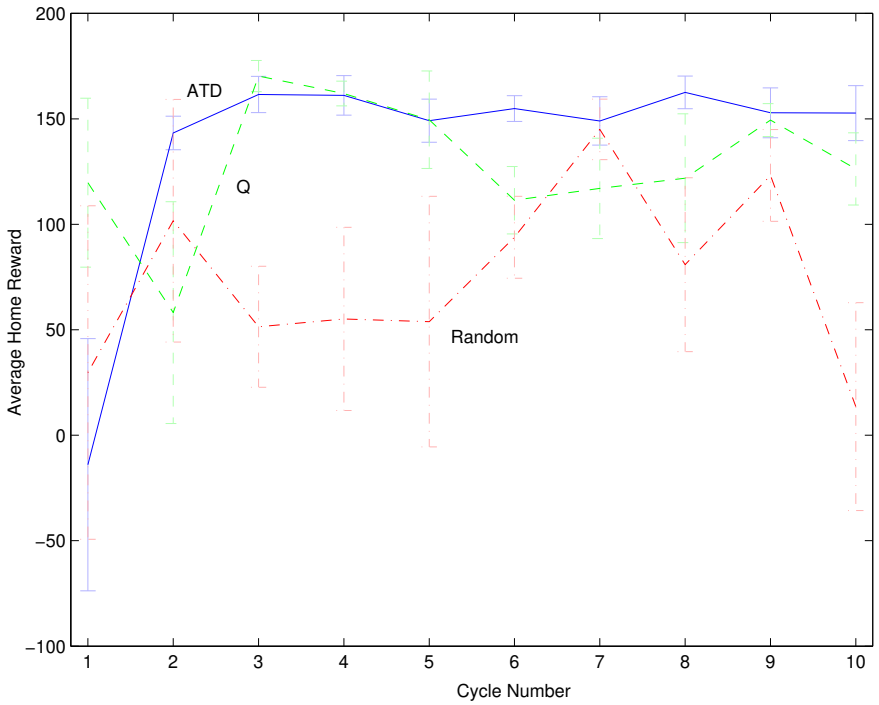


Figure 6.11: Average Home Reward over Time: The Second Arena

Both reinforcement learning methods perform better than the random model even through the

second arena was designed to make stumbling across the home area easy.

The reinforcement learning methods were thus able to cope with a noisy environment that underwent change during the agent's lifetime. The error bars in the second arena are slightly larger than those in the first, but this is likely to be a reflection of the noise inherent in the environment.

6.3.3 The Third Arena

The third arena was the most difficult arena faced by the robot, with the longest path to the puck and the most complex map. Due to time constraints, and because it had already been shown to perform poorly, no runs were performed with the random model. In addition, data from only five reinforcement learning model runs were used rather than seven[3].

Figure 6.12 shows the average puck reward over time for the third arena. It demonstrates decisively that the asynchronous algorithm outperforms the synchronous one when a long path to the goal must be constructed. The asynchronous algorithm consistently found and learned a short goal to the puck by the sixth cycle, whereas the synchronous algorithm did not manage to consistently find a good path at all.

The path commonly learned by the asynchronous model is shown on the left side of Figure 6.13. Even though this is a fairly complex arena, the robot manages to learn a direct path to the puck. A representative path for the synchronous model is shown on the right, and is clearly not as direct as the path learned by the synchronous model.

Figure 6.14 shows the preferred puck transitions for asynchronous and synchronous models. The map obtained from the asynchronous model (on the left) shows that the path to the puck has been propagated throughout the map, whereas it is clear from the map obtained from the synchronous model that the path to the puck has propagated slowly, with only the transitions very close to the puck transition having high values (indicated by dark arrows).

The difference between the two learning models is less pronounced in Figure 6.15, which shows the average home reward obtained by the two models over time. The asynchronous model again consistently finds a good solution quickly, at around the fourth cycle. The synchronous model

[3]Seven runs of both reinforcement learning models were actually performed, but once the arena had been deconstructed two of the Q-learning runs were discovered to have incorrect maps which could have detrimentally affected their performance.

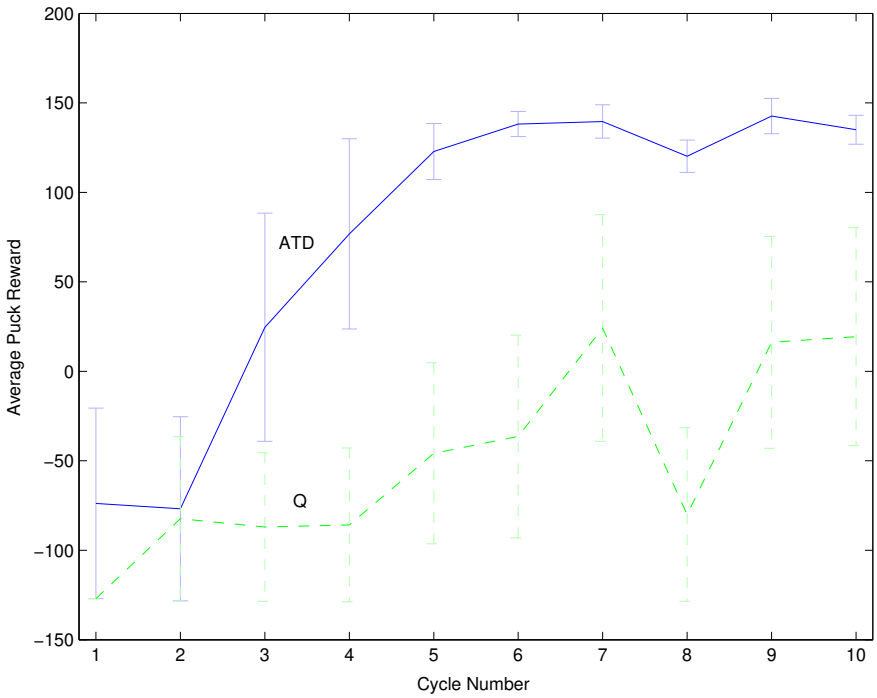


Figure 6.12: Average Puck Reward over Time: The Third Arena

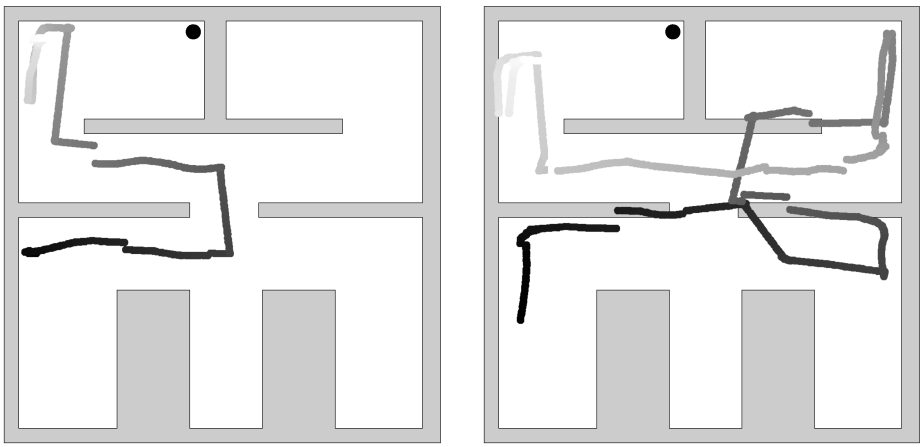


Figure 6.13: Learned Puck Routes in the Third Arena

takes longer, reaching the same conditions at around the seventh cycle, but still performs well.

A potential explanation for the difference in performance between the two models could be that since both models explore, and both must initially start all runs in the home area, the synchronous model would experience many transitions near the home area and thus be able

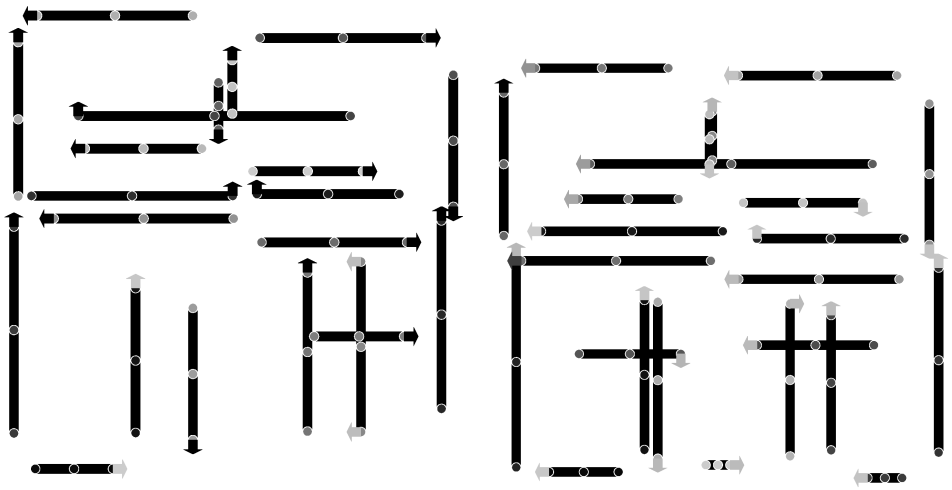


Figure 6.14: Preferred Puck Transitions Maps for the Third Arena

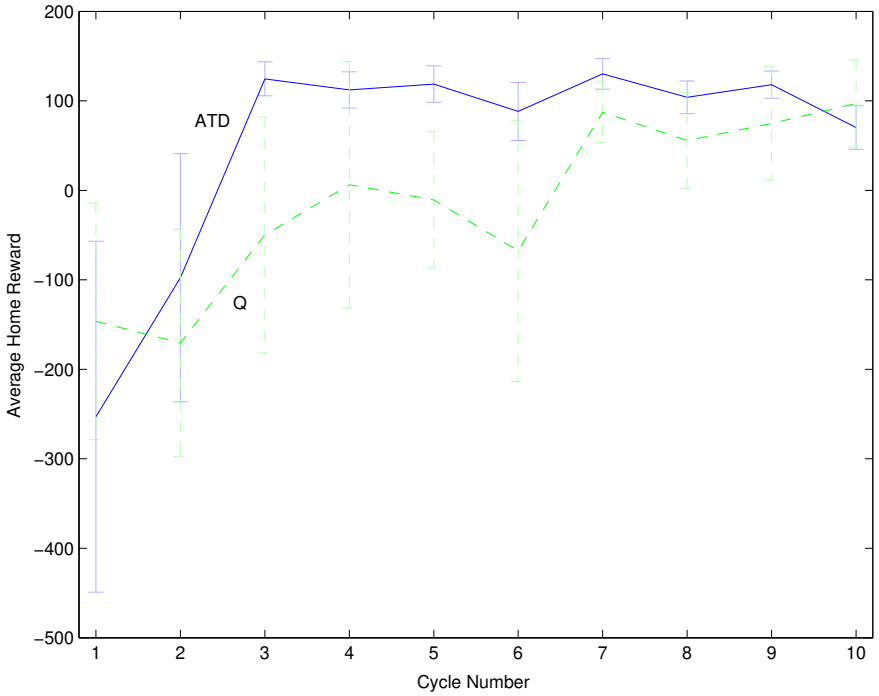


Figure 6.15: Average Home Reward over Time: The Third Arena

to build a path earlier than in the case of the puck, where it would be much less likely to experience a puck sighting repeatedly without having built a path to it first.

One revealing aspect of the robot type's behaviour was the apparent repetition of transitions

by the synchronous model, where several transition experiences were required to drive the optimistic initial transition values down. The asynchronous model repeated transitions only when it took several transition attempts to determine that an unexplored transition was illegal. This gave it more time to explore and allowed for wider coverage of the map, and may have contributed towards its superior performance.

6.3.4 Convergence

The final issue to be considered is that of convergence. In Chapter 3, the argument was made that the amount of time a situated agent takes to make a transition may be sufficient to allow an asynchronous reinforcement learning algorithm over a topological map to converge between transitions.

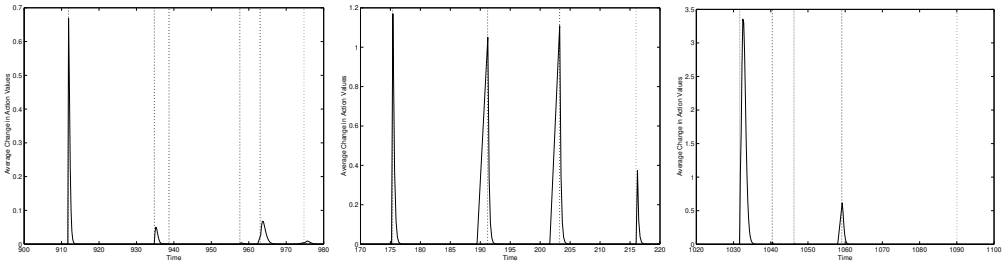


Figure 6.16: ATD Average Action Value Changes Over Time

Figure 6.16 contains three samples (one from each arena) showing the average change in action value over an 80 second slice of time, where the dotted vertical lines mark transition occurrences. Transitions function as event points, where a decision is required and where the reinforcement learning complex receives new data. The third graph is from a run in the third arena where the puck was discovered after 17 minutes, and the robot had built a nearly complete map of the arena. In all three graphs, the action values are disturbed by each event point but converge comfortably before the next one occurs.

Although the remainder of the data shows a similar pattern, convergence cannot be conclusively proved here because data could not be collected sufficiently rapidly to rule out the possibility of very small changes being made before some event points. For example, in the second graph in Figure 6.16, although it appears that some of the spikes begin before an event point, this is an artifact of the sampling rate of the event point data, as spikes can only ever fact start at event points.

6.3.5 Summary

It is clear from the results presented above that both reinforcement learning models perform better than the random movement strategy for this task, and that both are capable of learning to find the puck and return home again quickly. Although the synchronous method (Q-learning) performs roughly as well as the asynchronous method (ATD) when finding fairly short paths, the asynchronous model performs better when a long path must be learned, and recovers more quickly than the synchronous model when the environment changes. The results also suggest that the interplay of the exploration drive, the distributed map and the other drives may have subtle but important effects on the robot's overall performance. Finally, the data obtained from all of the runs suggests that the asynchronous model is able to converge between transitions, so that the choices made by the agent are optimal given the knowledge that it has.

6.4 Conclusion

This chapter has described the addition of two reinforcement learning models and a random choice model to Dangerous Beans, the map building robot developed in Chapter 5, and presented the results obtained by running these models in the experiment presented in Chapter 4. The results show that the model developed in Chapter 3 is feasible, and brings with it definite behavioural benefits for the puck foraging task.

Chapter 7

Discussion

7.1 Introduction

This chapter provides a discussion of the research presented in this dissertation and the research issues raised by it. The following section examines the significance of the model developed in Chapter 3, as well as the experiment constructed to test it and the results thus obtained. The limitations of both the model and the experimental design and implementation are then considered, followed by a discussion of their implications for both reinforcement learning and situated learning models in general. The final section concludes.

7.2 Significance

The model developed in this dissertation is significant because it has shown that the behaviour-based style of robot control architecture and reinforcement learning models can be fully integrated to produce an autonomous robot that is capable of rapid learning. This integration has the potential to widen the scope of behaviour-based systems and lead to the synthesis of robots that, like Dangerous Beans, learn from their environment in order to improve their performance. In addition, the use of a reinforcement learning model developed with the conditions faced by situated agents in mind has been shown to offer better performance than a model developed using the standard theoretical emphasis. This indicates that a similar change of emphasis may offer advantages when developing mobile robots using other learning models, and

that the integration of layered learning models and behaviour-based robotics is a promising research area.

7.3 Limitations

The work presented in this dissertation is not without its limitations. These can be divided into the limitations of the model presented in Chapter 3, and the limitations of the experiment and its implementation, as presented in chapters 4, 5 and 6.

One of the major limitations of the model is the fact that it relies upon a learned topological map. In some situations, it may not be possible to feasibly build or maintain such a map, and in others, the map may become prohibitively large. In cases where the map becomes large, the reinforcement learning layer on top of it is unlikely to be able to learn quickly enough to be useful in real time. Such situations may require the addition of other learning models, the use of hierarchical reinforcement learning methods (e.g., Digney (1998)), or one or more further topological mapping layers in order obtain a map that is small enough to be useful.

The other primary difficulty inherent in the model is that it inherits all of the limitations of reinforcement learning. Reinforcement learning is not appropriate for all types of problems, and imposes significant state property requirements. States must have the Markov property, which may be difficult to obtain in situated environments due to perceptual aliasing, where different states are perceptually indistinguishable. In such cases, the use of further internal state (Crook and Hayes, 2003) or active perception may be required in order to disambiguate states.

The experimental design and implementation also suffer from some limitations. The environment used in the experiment required significant engineering because of the robot's sensory limitations. This is not a major flaw because it did not serve to remove any of the fundamental difficulties inherent in the task. Nevertheless, the level of engineering required for the system as a whole was very high, with the distributed map building and dead reckoning correction in particular requiring a great deal of time and effort in order to function correctly.

Finally, the experimental results given in Chapter 6 are only the results of the application of the model in a single application area. Although these results suggest that the model is promising and may work in other domains, further experiments will be required in order to confirm this.

7.4 Implications

This section considers some of the implications of the research presented in this dissertation for situated reinforcement learning, and robot learning models in general. The following sections consider the implications for situated reinforcement learning, planning behaviour, layered learning and emergent representations in turn, with the last section providing a highly speculative discussion of the role of learning models in the study of representation.

7.4.1 Situated Reinforcement Learning

The results presented in this dissertation have shown that a learning algorithm specifically geared towards the problems faced by situated agents can outperform one developed with a more traditional theoretical approach in mind. A situated agent employing a learning algorithm is required to learn in real time, using a reasonable amount of computation, in a manner that provides behavioural benefits. Although such agents have the benefit of being able to harness the parallelism inherent in their distributed control architectures, the evolution of extremely complex methods to make learning strategies feasible is not plausible, especially when the naive implementations of such methods provide no behavioural benefit whatsoever. For example, although the behaviour generated by the model in Toombs *et al.* (1998) corresponds closely to that exhibited by gerbils under experimental conditions, very few gerbils are likely to survive the fifty thousand training instances required to obtain convergence using their model.

A great deal of work has gone into making reinforcement learning using a single control process over very large state spaces feasible (e.g., Smart and Kaelbling (2000)). The results obtained in this dissertation suggest that perhaps more effort should go into the development of methods (such as layering reinforcement learning over a topological map) that create conditions under which learning is feasible. The two most promising approaches to this are the integration of a priori knowledge and learning bias into learning models (Bryson, 2002) and the use of layered learning models (Stone and Veloso, 2000). Situated learning clearly merits further study as a subfield of machine learning in its own right.

7.4.2 Planning Behaviour

One of the original criticisms of behaviour-based robotics was that systems built using it would never be able to plan, because its emphasis on distributed control, reactive behaviour and a lack of syntactic representations preclude the use of traditional planning algorithms (Brooks, 1987).

Although the situated view of intelligent agents does not consider the construction and execution of plans to be the primary activity performed by an intelligent agent in the same way that classical artificial intelligence does (Agre and Chapman, 1990; Brooks, 1987), the generation of some form of planning behaviour is still an important aspect of intelligent behaviour. Reinforcement learning with the use of a model can, however, be viewed as a form of plan learning (Sutton and Barto, 1998), where optimal choices can be made using purely local decisions once the state or state-action value table has converged. The results in Chapter 6 show that Dangerous Beans can be said to be displaying planning behaviour, because it finds the optimal path to the puck and back to its home area given the knowledge that it has, without the use of a planner in the classical sense of the word.

The research in this dissertation has therefore shown that, in at least some cases, planning behaviour can be generated using reinforcement learning methods, and does not necessarily require traditional planning.

7.4.3 Layered Learning

Layering reinforcement learning on top of a topological map is one instance of the layered learning approach introduced by Stone (2000). The layering of learning models is a powerful and general idea that has not yet been fully explored.

One of the implications of this research is that a kind of feedback between layered learning systems is possible, where the performance of each algorithm biases the other's learning opportunities. Since most machine learning research has concentrated on only one learning model in isolation from all the others, there is significant scope for future research into the kinds of biases that two interacting learning algorithms can impose on each other.

Another interesting aspect of using one type of learning to make another feasible is that it suggests an information requirement ordering for learning models, and perhaps a similar evolutionary ordering where species must evolve some types of learning models before others in

order to obtain behavioural benefits. For example, once Dangerous Beans was capable of distributed map learning, the addition of a reinforcement learning layer provided it with significant behavioural benefits but required far less engineering effort than that required to develop the map learning layer in the first place. There may be scope for the investigation of these kinds of relationships between learning models through artificial evolution (Harvey, 1995).

Layered learning models may also have interesting implications in terms of emergent behaviour. Since the interaction of multiple control processes and a complex environment results in complex behaviour, it is reasonable to expect that the interaction of multiple learning models, multiple control processes, and a complex environment will likewise result in complex learning behaviour.

7.4.4 Emergent Representations

The behaviour-based approach to artificial intelligence has caused a change in the way that many artificial intelligence researchers view behaviour. Behaviour is now considered to be the emergent result of the complex interaction between an agent's multiple control processes, morphology, and environment. However, there has been no corresponding change in the way that researchers view representation. Some behaviour-based roboticists simply ignore it, while others maintain the classical view that representation is the result of an internal syntactic symbol system.

Simply ignoring the role of representation in intelligent behaviour is not a tenable position. The real question is not whether or not representation must exist in an intelligent agent, but rather *in what form* and *in what manner* it exists. Although Brooks (1991a) is often considered an argument against any form of representation whatsoever[1], it is actually an argument against the use of central syntactic representational systems for control. In fact, Brooks (1991a) claims that useful representations should be distributed and emergent, and that such things would sufficiently different from what is traditionally considered a representation that they should be called something else.

One powerful way to study representations in situated agents is through learning. Representations themselves do not offer an agent a behavioural advantage – rather, the simple fact that anything an agent wishes to represent must first be learned implies that the types of learning

[1]This paper is, after all, entitled "Intelligence without Representation".

models used by the agent dictate its representations and the behavioural advantages it receives.

This imposes a dual constraint on the types of representations an agent will find beneficial: the relevant learning model must be feasible in that it must be able to learn in real time, and the behaviour it generates must be useful, in that it provides behavioural advantages appropriate to the agent's level of competence. Since situated learning is only feasible when it is task-specific, it follows that such representations are also likely to be task specific. In the same way that there is never likely to be a single general purpose tractable learning algorithm, there is no known representational system that is capable of handling a wide spectrum of knowledge at different levels of detail without becoming computationally intractable. Situated intelligence should be expected to develop in such a way as to facilitate cheap computation and rapid learning, whenever possible.

Given the assertion that representation arises through the presence of learning models, it follows that representations must form from the structures generated by those learning models and their interaction with each other. These representations would be emergent in the sense that they would not be composed of atomic, syntactic symbols, but would instead be complex entities formed by the association of several task-specific structures at different levels of detail, organised into a loosely hierarchical distributed structure. Such complexes may only be identifiable as symbols given a particular linking context.

For example, when observing the behaviour of Dangerous Beans, an outside observer would say that the robot has a representation of a map, and a representation of the path to the puck. However, Dangerous Beans has a distributed map, which is an emergent structure to some extent – it exists because of the behaviour and interaction of all of the behavioural modules that make it up with the environment and each other. Similarly, nowhere in Dangerous Beans' control structure is there a path representation. Rather, there is a reinforcement learning complex, another emergent structure, that consists of a set of values that cause the agent to make certain choices when it is at certain places. The path is an emergent property of these choices, and it results from the interaction of the distributed map, the reinforcement learning complex embedded in it, and the environment itself. The claim that some representation of the path to the puck exists outside of the interaction of these elements is simply false.

7.5 Conclusion

Although some of the ideas expressed in this chapter are highly speculative, the research presented in this dissertation has, despite its limitations, provided some motivation for the study of behaviour-based reinforcement learning and situated learning generally, and raised interesting questions about the nature of representation and its relationship to learning.

Chapter 8

Conclusion

8.1 Introduction

This chapter concludes this dissertation by providing an overview of the research contributions contained in it, their significance, and the areas in which the research presented here could be extended in the future.

8.2 Contribution

The contribution of this dissertation is threefold. First, it has introduced a model that integrates the behaviour-based style of robot architecture and reinforcement learning. Second, it has detailed the development of a mobile robot that uses this model to learn to solve a difficult problem in the real world, providing both an engineering contribution, and resulting in data that supports the claim that the model is capable of learning in real time, in the real world. Finally, through the development of a fully behaviour-based layered learning system, some progress has been made towards bringing these two powerful and important ideas together.

8.3 Significance

The learning model and robot described in this dissertation have displayed rapid learning and intelligent behaviour in a challenging real world environment while retaining full autonomy

through the integration of behaviour-based control and reinforcement learning. Although this is significant in itself, it has more importantly provided a basis for further investigation into behaviour-based reinforcement learning and its potential for generating complex, goal-directed behaviour through rapid learning.

Furthermore, the research described here suggests that layering learning models geared specifically towards the problems faced by situated agents, and coupling them with behaviour-based methods, is a promising approach to the development of robots that exhibit complex, adaptive behaviour. This research has therefore gone some way towards making the case for the further fusion of learning models and behaviour-based control.

8.4 Future Work

The research presented in this dissertation is by no means conclusive, and could be extended in at least three directions. First, the reinforcement learning model presented here could be further developed. Future work could involve ways to sensibly handle the competing demands of multiple reinforcement learning complexes, the development of more efficient algorithms, theoretical results specific to the problems posed by situated learning, and the inclusion of a priori knowledge of the problem space in a principled way.

Second, the model presented here could be applied in other domains in order to verify whether or not it is indeed applicable outside of landmark-based navigation. The other two examples given in Chapter 3 may be good places for this to start.

Finally, the integration of other types of learning into the reinforcement complex could be considered. For example, social imitation could be used to seed the state space when it is too large for a solution to be found in a practical amount of time, associative learning methods could be used to create more useful reward functions, and so forth. The ultimate goal of such an investigation would be a classification of all known learning methods in terms of their information requirements and feasibility conditions, and a set of guidelines for their use in developing complex learning systems for autonomous agents.

8.5 Conclusion

If behaviour-based systems are to have any hope of moving beyond insect-level intelligence, they must begin to incorporate learning mechanisms at all levels of behaviour. This dissertation has argued that this is more than just a matter of inserting learning models into behaviour-based systems – it is a matter of understanding what is required to make learning feasible in the real world, how to layer learning models so that their interaction facilitates the generation of complex behaviour, and how to truly integrate learning into behaviour-based systems. The research presented here represents a small but hopefully concrete step in that direction.

Bibliography

Agre, P. and Chapman, D. (1990). What are plans for? In P. Maes, editor, *New Architectures for Autonomous Agents: Task-level decomposition and Emergent Functionality*. MIT Press, Cambridge, MA.

Arkin, R. (1998). *Behavior-Based Robotics*. MIT Press, Cambridge, Massachusetts.

Ballard, D. and Brown, C. (1982). *Computer Vision*. Prentice Hall.

Braitenberg, V. (1984). *Vehicles – Experiments in Synthetic Psychology*. MIT Press, Cambridge, Massachusetts.

Brooks, R. (1987). Planning is just a way of avoiding figuring out what to do next. In R. Brooks, editor, *Cambrian Intelligence : the early history of the new AI*, pages 103–110. The MIT Press, Cambridge, Massachusetts.

Brooks, R. (1991a). Intelligence without representation. In J. Haugeland, editor, *Mind Design II*, pages 395–420. MIT Press, Cambridge, Massachusetts.

Brooks, R. (1991b). The role of learning in autonomous robots. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory (COLT '91)*, pages 5–10, Santa Cruz, CA.

Bryson, J. (2002). Modularity and specialized learning: Reexamining behavior-based artificial intelligence. In *Proceedings of the Workshop on Adaptive Behavior in Anticipatory Learning Systems*. Springer.

Choset, H. and Nagatani, K. (2001). Topological simultaneous localization and mapping (SLAM): Towards exact localization without explicit localization. *IEEE Transactions on Robotics and Automation*, **17**(2).

Crook, P. and Hayes, G. (2003). Learning in a state of confusion: Perceptual aliasing in grid world navigation. In *Proceedings of the 4th British Conference on (Mobile) Robotics: Towards Intelligent Mobile Robots (TIMR 2003)*.

Digney, B. (1998). Learning hierarchical control structures for multiple tasks and changing environments. In R. Pfeifer, B. Blumberg, J. Meyer, and S. Wilson, editors, *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, Zurich, Switzerland.

Dudek, G., Freedman, P., and Hadjres, S. (1993). Using local information in a non-local way for mapping graph-like worlds. In *Proceedings of the International Joint Conference of Artificial Intelligence*, Chambery, France.

Ferrel, C. (1996). Orientation behavior using registered topographic maps. In P. Maes, M. Matarić, J. Meyer, J.Pollack, and S. Wilson, editors, *From Animals to Animats 4: Proceedings of the Fourth International Conference on the Simulation of Adaptive Behavior*, Cape Cod, MA.

Fritzke, B. (1995). A growing neural gas networks learns topologies. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7*. MIT Press, Cambridge, MA, USA.

Harvey, I. (1995). *The Artificial Evolution of Adaptive Behaviour*. DPhil thesis, School of Cognitive and Computing Sciences, University of Sussex.

Humphrys, M. (1996). Action selection methods using reinforcement learning. In *From Animals to Animats 4: the Fourth International Conference on the Simulation of Adaptive Behaviour (SAB-96)*, Cape Cod, MA, USA. MIT Press.

K-Team SA (1999a). *Khepera K213 Vision Turret User Manual*. Lausanne, Switzerland.

K-Team SA (1999b). *Khepera User Manual*. Lausanne, Switzerland.

Kohonen, T. (1989). *Self-Organization and Associative Memory*. Springer-Verlag, 3rd edition.

Lucas, G. (2000). *A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators*. `http://rossum.sourceforge.net/papers/DiffSteer`, The Rossum Project.

Maes, P. and Brooks, R. (1990). Learning to coordinate behaviors. In *Proceedings of the American Association of Artificial Intelligence*, Boston, MA.

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

Marsland, S., Shapiro, J., and Nehmzow, U. (2002). A self-organising network that grows when required. *Neural Networks*, **15**(8–9), 1041–1058.

Matarić, M. (1990). *A Distributed Model for Mobile Robot Environment-Learning and Navigation*. Master's thesis, MIT Artificial Intelligence Laboratory.

Matarić, M. (1994). Reward functions for accelerated learning. In *International Conference on Machine Learning*, pages 181–189.

Matarić, M. and Brooks, R. (1990). Learning a distributed map representation based on navigation behaviors. In R. Brooks, editor, *Cambrian Intelligence : the early history of the new AI*. The MIT Press, Cambridge, Massachusetts.

Mitchell, T. (1997). *Machine Learning*. McGraw-Hull.

Morén, J. (1998). Dynamic action sequences in reinforcement learning. In R. Pfeifer, B. Blumberg, J. Meyer, and S. Wilson, editors, *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, Zurich, Switzerland.

Murphy, R. (2000). *Introduction to AI Robotics*. MIT Press, Cambridge, Massachusetts, 1st edition.

Pfeifer, R. and Scheier, C. (1999). *Understanding Intelligence*. MIT Press, Cambridge, MA.

Schmolke, A. and Mallot, H. (2002). Polarization compass for robot navigation. In *The Fifth German Workshop on Artificial Life*, pages 163–167.

Smart, W. and Kaelbling, L. (2000). Practical reinforcement learning in continuous spaces. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 903–910.

Smith, A. J. (2002). Applications of the self-organising map to reinforcement learning. *Neural Networks*, **15**, 1107–1124.

Stone, P. (2000). *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*. MIT Press.

Stone, P. and Veloso, M. (2000). Layered learning. In *Proceedings of the 11th European Conference on Machine Learning*, pages 369–381, Barcelona, Spain. Springer, Berlin.

Sutton, R. (1990). Reinforcement learning architectures for animats. In J. Meyer and S. Wilson, editors, *From Animals to Animats: Proceedings of the International Conference on Simulation of Adaptive Behavior*.

Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

Toombs, S., Phillips, W., and Smith, L. (1998). Reinforcement landmark learning. In R. Pfeifer, B. Blumberg, J. Meyer, and S. Wilson, editors, *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, Zurich, Switzerland.

Watkins, C. and Dayan, P. (1992). Q-learning. *Machine Learning*, **8**, 279–292.

Whiteson, S. and Stone, P. (2003). Concurrent layered learning. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems*.

Phelile.