$\mathcal{Q}$-Learning for Robot Control

A thesis submitted for the degree of
Doctor of Philosophy
of The Australian National University.

Chris Gaskett
Bachelor of Computer Systems Engineering H1 (RMIT University)
Bachelor of Computer Science (RMIT University)

Supervisor:
Professor Alexander Zelinsky

2002

Statement of Originality

These doctoral studies were conducted under the supervision of Professor Alexander Zelinsky. The work submitted in this thesis is a result of original research carried out by myself, in collaboration with others, while enrolled as a PhD student in the Department of Systems Engineering at the Australian National University. It has not been submitted for any other degree or award.

Chris Gaskett

Abstract

$\mathcal{Q}$-Learning is a method for solving reinforcement learning problems. Reinforcement learning problems require improvement of behaviour based on received rewards. $\mathcal{Q}$-Learning has the potential to reduce robot programming effort and increase the range of robot abilities. However, most current $\mathcal{Q}$-learning systems are not suitable for robotics problems: they treat continuous variables, for example speeds or positions, as discretised values. Discretisation does not allow smooth control and does not fully exploit sensed information. A practical algorithm must also cope with real-time constraints, sensing and actuation delays, and incorrect sensor data.

This research describes an algorithm that deals with continuous state and action variables without discretising. The algorithm is evaluated with vision-based mobile robot and active head gaze control tasks. As well as learning the basic control tasks, the algorithm learns to compensate for delays in sensing and actuation by predicting the behaviour of its environment. Although the learned dynamic model is implicit in the controller, it is possible to extract some aspects of the model. The extracted models are compared to theoretically derived models of environment behaviour.

The difficulty of working with robots motivates development of methods that reduce experimentation time. This research exploits $\mathcal{Q}$-learning's ability to learn by passively observing the robot's actions—rather than necessarily controlling the robot. This is a valuable tool for shortening the duration of learning experiments.

Acknowledgements

I thank my supervisor, Professor Alexander Zelinsky, and my previous supervisor Dr David Wettergreen, for guiding me through this research. Thank you to all the staff and students of Systems Engineering, past and present, for their constant support and friendship. Thanks also to all of the visitors to the Robotic Systems Laboratory for their valuable advice.

The mobile robot software was built upon the work of Dr Gordon Cheng (1996), and used correlation software developed by Dr Jochen Heinzmann (1997). Luke Fletcher ported Dr Cheng's software to the Nomad, integrated the learning system into the software, participated in the experiments, helped with video editing, and co-authored two papers. The active head software and experiments were the work of myself and Peter Brown. Vision libraries were provided by Dr Sebastien Rougeaux. Orson Sutherland, Leanne Matuszyk, and Harley Truong provided active head system software. Dr Thomas Brinsmead and Peter Brown assisted with the theoretical pendulum model. Dr Simon Thompson assisted with video editing. Karen Montefiore, Gareth Loy, Luke Fletcher, and Dr Simon Thompson assisted with preparation of diagrams. Dr Peter Bartlett and Dr Jonathan Baxter gave valuable advice about learning systems theory. Dr Baxter also supplied some example feedforward neural network source code. Professor Brian Anderson, Professor John Moore, Professor Robert Bitmead, Dr Thomas Brinsmead, and Dr Thomas Moor answered my questions about control systems. The display software for the cart-pole task was based on Professor Charles Anderson's (1998) pole balancing computer game. Dr Gordon Cheng reviewed an earlier draft. James Ashton kept the computers' wheels turning. Jenni Watkins, Marita Rendina, Karen Montefiore, and Rosemary Shepherd arranged everything and provided moral support. Dr Jochen Heinzmann, Dr David Jung, Dr Gordon Cheng, Dr Rochelle O'Hagan, and Andrew Brooks made me feel welcome from the start. My friends from RMIT University taught me the value of teamwork done well.

My family gave me unflagging support throughout the preparation of this thesis. Finally, I thank my wife Rika, for everything.

Chris Gaskett

List of Publications

Gaskett, C., Brown, P., Cheng, G., and Zelinsky, A. (2003), Learning implicit models during target pursuit, *in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA2003)*, Taiwan.

Gaskett, C., Fletcher, L., and Zelinsky, A. (2000a), Reinforcement learning for a vision based mobile robot, *in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2000)*, Takamatsu, Japan.

Gaskett, C., Fletcher, L., and Zelinsky, A. (2000b), Reinforcement learning for visual servoing of a mobile robot, *in Proceedings of the Australian Conference on Robotics and Automation (ACRA2000)*, Melbourne, Australia.

Gaskett, C., Wettergreen, D., and Zelinsky, A. (1999a), Q-learning in continuous state and action spaces, *in Proceedings of the 12th Australian Joint Conference on Artificial Intelligence*, Sydney, Australia.

Gaskett, C., Wettergreen, D., and Zelinsky, A. (1999b), Reinforcement learning applied to the control of an autonomous underwater vehicle, *in Proceedings of the Australian Conference on Robotics and Automation (AuCRA'99)*, Brisbane, Australia.

Wettergreen, D., Gaskett, C., and Zelinsky, A. (1998), Development of a visually-guided autonomous underwater vehicle, *in Proceedings of OCEANS98*, IEEE, Nice.

Wettergreen, D., Gaskett, C., and Zelinsky, A. (1999a), Autonomous control and guidance for an underwater robotic vehicle, *in Proceedings of the International Conference on Field and Service Robotics (FSR'99)*, Pittsburgh, USA.

Wettergreen, D., Gaskett, C., and Zelinsky, A. (1999b), Reinforcement learning for a visually-guided autonomous underwater vehicle, *in Proceedings of the International Symposium on Unmanned Untethered Submersibles Technology (UUST'99)*, Durham, New Hampshire, USA.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A monk asked, "What is the mind that the patriarch bought from the West?"

The master got up from his seat.

The monk said, "Is it nothing more than this?"

The master said, "I haven't said anything yet.

Joshu, c. 890, no. 53[1]

The fundamental task in robotics is to create an intelligent machine which achieves a desired task through interaction with the world. The robot's behaviour is reasoning, acting and reacting based on knowledge and sensed information. Flexible behaviour requires the ability to learn—to acquire knowledge, or improve skills, based on experience, observation, or instruction.

Learning and robotics is an attractive combination. Our own intelligent behaviour includes the ability to change our behaviour based on various forms of feedback. In robotics we wish to emulate this adaptability in the hope of being able to achieve a broader range of tasks in a more robust way. The idea has inspired various learning systems for robots. However, implementing these systems has shown the difficulty of applying current learning algorithms to robotics problems. To be practical, learning algorithms need to become more robust to the effects of real sensors, real actuators, and real time. The purpose of our research is to work towards a reinforcement learning approach that is appropriate for robotics.

[1]The quotations introducing each chapter are from a Ch'an Buddhist master from China: Chao-chou Ts'ung-shen. He is better known to English speakers as Joshu Jushin, the Japanese version of his name. The translation is by James Green.

1.1 Reinforcement Learning for Robotics

Reinforcement learning allows a robot to synthesise and improve behaviour through trial and error. Within the reinforcement learning framework a learning task is defined by a reward function: an appropriate choice of action for the current state results in rewards, reinforcing the helpful behaviour. Conversely, an inappropriate choice of action results in more negative rewards, inhibiting the non-helpful behaviour. Further, the long-term results of actions must be considered. The reinforcement learning framework includes the concept of cumulative rewards over time so that a robot can learn to choose actions that promote positive long-term results.

The reinforcement learning system's information is based on information sensed by the robot. The reward signal is also based on sensed information. The learning system's actions influence the robot's actuators. The learning system must be designed appropriately to deal with the robot's configuration and task. Learning algorithms cannot assume that global, reliable information is available. Sensors merely measure some property of the environment; they do not create a perfect internal model that can be manipulated to predict how world will change if various actions are performed. The robot's environment changes in real-time, and changes while the robot is choosing which action to perform. Many robot sensors report continuously variable results, such as positions and velocities. Many actuators also respond to continuous values. However, the world of discourse for most reinforcement learning algorithms is a discrete representation. Discretising the mostly continuous world so that discrete reinforcement learning can be applied creates problems, including information loss and coarseness of control.

Simulations tend to hide the main difficulty in robotics: dealing with the environment. Valuable data can be obtained from simulations, but to interpret them as a simulation of a robotics problem is misleading. Algorithms for robots must be tested on robots.

1.2 Research Objectives

The purpose of our work is to increase the usefulness of reinforcement learning techniques for robotics. Proposed methods are to be validated using real robots. The problem of reinforcement learning with continuous states and actions is the main focus of investigation. The time and safety constraints imposed by robotics problems produces a secondary focus: flexible learning and efficient use of measured data. Other aims include identifying directions for further research.

1.3 Contributions

A reinforcement learning method for continuous state and action problems is presented: *Wire Fitted Neural Network Q-Learning*. The major application of the algorithm is uncalibrated control of non-linear, multiple input, multiple output systems. Appropriate systems include mobile robots, active heads, robotic arms, and legged robots. Appropriate control tasks are midway between logical, multiple step problems like chess, and simple control tasks that could be solved adequately through PID control.

The wire fitted neural network method is evaluated using two robots: a mobile robot, and a binocular active head. The mobile robot learns to pursue objects and explore using vision. Smooth movements are achieved.

A binocular active head is a pair of cameras motorised to imitate human vision including head and eye movements. In this work an active head learns to use vision to accurately pursue objects. The learned controller performs lag-free tracking of a swinging target. The controller achieves this by implicitly predicting the target's behaviour. The controller is the first continuous state or continuous state and action reinforcement learning controller for an active head. A novel mechanism is used to aid in producing training data for the learning algorithm. The mechanism generates a mixture of smoothly moving and instantaneously moving visual targets.

Continuous state and action reinforcement learning is problematic since discrete reinforcement learning's guarantee of convergence no longer applies. Several divergence pitfalls are discussed and partial solutions are proposed.

The passive learning approach is proposed as a practical path for future research. Passive learning systems learn from observing the behaviour of existing controllers or through recycling data. The approach recognises the high cost of gathering data for learning and adds flexibility to the learning process. The proposed passive learning approach can refine or combine existing controllers. It is applicable to both discrete and continuous problems.

1.4 Thesis Organisation

Chapter 2 introduces reinforcement learning. It focuses on the difficulties in applying reinforcement learning to continuous state and action problems and describes existing approaches. The concept of passive learning is also introduced.

Chapter 3 describes the wire fitted neural network approach to continuous state and action reinforcement learning. The algorithm is applied to a simulation problem. Design of reward functions to encourage smooth control is discussed.

Chapter 4 investigates the performance of the algorithm in simulation, describing enhancements and methods for choosing parameters. Convergence problems are discussed and some partial solutions are proposed.

Chapter 5 discusses the difficulty of moving the algorithm from simulation problems to robotics problems. This includes unreliable sensor data, real-time issues, and safety management. Suitable methods are proposed including multithreaded and client/server architectures.

Chapter 6 applies the algorithm to two mobile robotics problems: visual servoing, and vision-based wandering. Passive learning techniques are demonstrated.

Chapter 7 applies the algorithm to control of a binocular active head. The learned controller is shown to be implicitly predicting the motion of a target. The disadvantages of pure model-free control are discussed.

Chapter 8 concludes with an assessment of the results and proposes directions for further investigation.

Chapter 2

A Review of Reinforcement Learning

A monk asked, "What is 'untutored wisdom'?"
The master said, "I have never taught you."

Joshu, c. 890, no. 263

Reinforcement learning allows a robot to synthesise and improve behaviour through trial and error—the robot must learn to act in a way that brings rewards over time. It is difficult to simulate or model a robot's interaction with its environment. Therefore, it is appropriate to consider model-free approaches to learning. As a real robot must deal with real quantities, positions and velocities, learning systems that only cope with discrete data are inappropriate. This chapter introduces reinforcement learning and summarises the important issues for robotics. A survey of reinforcement learning algorithms that cope with continuously variable information and continuously controllable actuators is also presented.

2.1 Reinforcement Learning

In reinforcement learning tasks the learning system must discover by trial-and-error which *actions* are most valuable in particular *states*. For an introduction to reinforcement learning refer to the work of Sutton and Barto (1998) and Kaelbling et al. (1996).

In reinforcement learning nomenclature the *state* is a representation of the current situation of the learning system's environment. The *action* is an output from the learning system that can influence its environment. The learning system's choice of actions in response to states is called its *policy*.

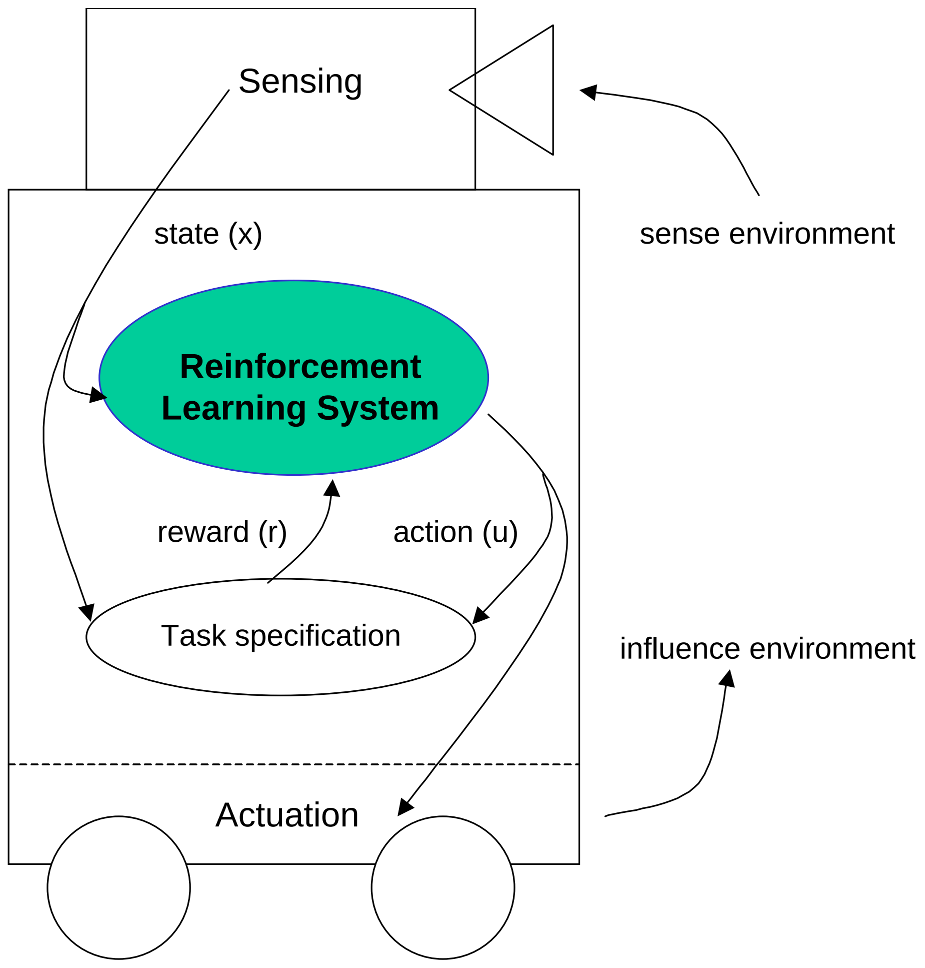


Figure 2.1: A basic architecture for robot control through reinforcement learning

Reinforcement learning lies between the extremes of supervised learning, where the policy is taught by an expert, and unsupervised learning, where there is a no evaluative feedback. In reinforcement learning evaluative feedback is provided in the form of a scalar *reward* signal that may be delayed. The reward signal is defined in relation to the task to be achieved; reward is given when the system is successfully achieving the task.

Figure 2.1 shows the interaction between the environment and the learning system in reinforcement learning applied to robot control. The interaction between states and actions is usually regarded as occurring in a discrete time Markov environment: the probability distribution of the *next-state* is affected only by the execution of the *action* in the current *state*.

The task specification block indicates the calculation of reward for a particular combination of state, action, and next-state. The user of the learning system must design the reward function to suit the task at hand. The reward signal can be made more positive to encourage behaviour or more negative to discourage behaviour.

The learning system's purpose is to find a policy that maximises the discounted sum of expected future rewards,

$$\text{value} = E\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots\right] \tag{2.1}$$

where E is the expectation operator; r is the reward; and γ is the discount factor, between 0 and 1. The discount factor makes rewards that are earned later exponentially less valuable. Reinforcement learning problems are multiple-step optimisation problems.

In reinforcement learning tasks the reward defines the objective. A poorly chosen reward function can cause the learning system not to converge, or to converge to a policy that does not achieve the desired task.

The degree of delay in the reward signal classifies reinforcement learning problems into three classes:

1. immediate reward;
2. delayed reward; and
3. pure-delayed reward.

In *immediate* reward problems the reward immediately and completely describes the value of taking an action in a particular state. Playing the card game *snap* is an example of an immediate reward problem. In snap, when a player notices that two successive cards of the same value have been laid down the player shouts 'snap' and slaps the pile of cards with their hand. In making the decision whether to 'snap' or not, only the current state of the cards is important; there is no need to plan for the future. In immediate reward problems there is no need to carry rewards backwards from the future. Therefore, these problems should be solved with γ (the discount factor) set to zero. Immediate reward problems are rare since actions usually take time to affect the environment.

Delayed reward problems span a wider range of tasks. In this class of problems there may be a reward available at every time step; however, it is not a complete evaluation of an action in a particular state. Playing the stock market is an example of a delayed reward problem: some actions yield immediate monetary rewards or losses, but decisions should also take the future into account. Robotics tasks are often delayed reward tasks. It is necessary to propagate rewards back through time; therefore, γ must be greater than zero.

In *pure-delayed* reward problems the reward for every step of the trial is the same, except possibly the last step, when a terminal reward is given that indicates success or failure. Playing backgammon is an example of a task that can be approached as a pure-delayed reward problem (Tesauro, 1992). In backgammon, success or failure occurs only at the end of the game. Rewards could be given for achieving subgoals; however, a pure-delayed reward approach ensures that the learning system will not learn to achieve the subgoals without achieving the main goal of winning the game. Solving pure-delayed reward problems generally requires γ close to one so that the reward can be carried back over many steps.

Although pure-delayed reward problems are the most studied class of reinforcement learning problems, they are not the most realistic model of real world tasks: in real world tasks there is usually some partial evaluation, or progress estimation, available before the end of the trial (Mataric, 1994). Treating all tasks as pure-delayed reward problems makes learning impractically slow.

The choice of reward function and state representation have been shown to be a major influence on the quality and timeliness of solutions found by reinforcement learning systems. An example is Wyatt et al.'s (1998) work on a robot box pushing task. The desired behaviour was to find boxes and push them to the walls of the room, the same task described by Connell and Mahadevan (1993a). Seemingly minor errors in the reward and state representation led to unexpected *box-avoider* and *wall-pusher* behaviours: the *box-finder* became a *box-avoider* because more reward was given for finding boxes than contacting the boxes; the *box-pusher* became a *wall-pusher* because reward was given immediately after collision with a wall due to sensor delays.

2.2 Reinforcement Learning for Robotics

The reinforcement learning paradigm is attractive for robotics researchers since it leads to the idea that a robot can learn from scratch to carry out complex tasks without calibration or programming. However, achievements to date have been unspectacular.

Complex robotics tasks have been achieved through reinforcement learning only by splitting them into simpler tasks. Connell and Mahadevan's (1993a) work is a well-known example. The task of pushing boxes across a room with a mobile robot was achieved by separating the task into several sub-behaviours. The sub-behaviours were learnt; coordination between the behaviours was hard-wired. Mataric (1994) demonstrated the complimentary problem of learned coordination of hard-wired behaviours with a multiple robot foraging problem.

Long learning times are a major problem for robotics due to limited hardware lifetime and reliability, safety, and monitoring requirements. Slow robot learning could be eliminated through learning in simulation—if adequate simulation were possible. Learning in simulation is simpler than learning in the real world. So much simpler that it led to the observation that "simulations are doomed to succeed" (Brooks and Mataric, 1993).

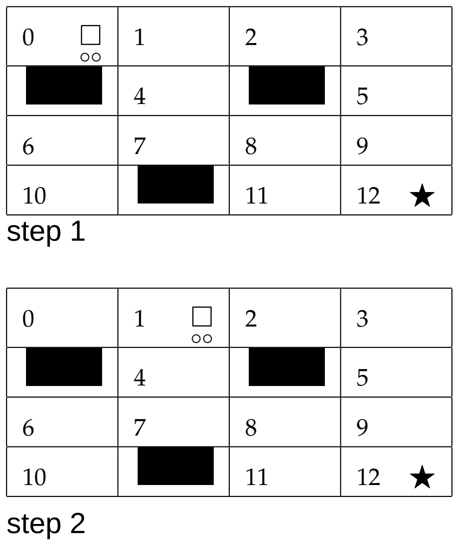
2.2.1 Simulation versus Reality

Brooks and Mataric (1993) described how simulations inevitably move toward the abilities of the algorithm under investigation, instead of the real problem. Simulation is rarely sufficiently realistic, and it is difficult to separate artifacts of the simulation from learning issues.

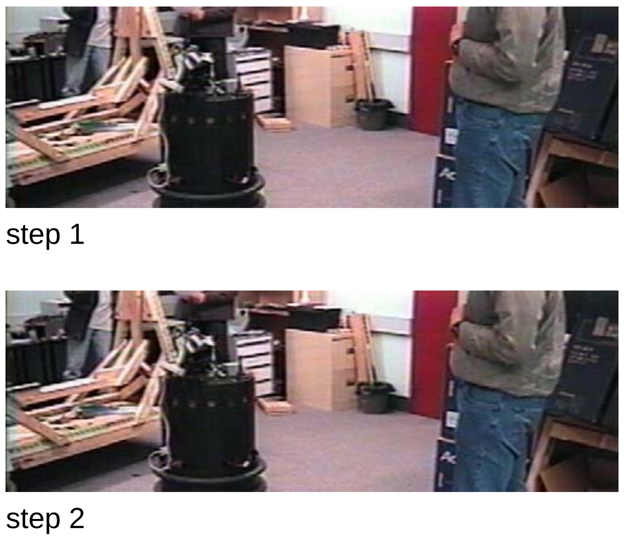
The main difficulty in robotics is interaction with the environment. A reinforcement learning system's state and reward information ultimately comes from the robot's sensors. Sensors merely roughly measure some property of the environment; they do not create a perfect internal model that can be manipulated to predict how the world will change if various actions are performed. It is inappropriate to model sensing errors by simply adding noise. Actuators do not perform high-level actions, for example moving a robot into a different room. The robot's environment changes in real-time, and changes while the robot is processing sensor data and deciding which action to perform. Consequently, simulations often allow the robot access to global data that could not possibly be available to the real robot.

The 'Grid World' simulation experiments often used to evaluate reinforcement learning algorithms are particularly far removed from the practical problems of robotics. Example grid world and real world tasks are shown in Figure 2.2. Grid world simulations often assume that the robot's location is known (as a grid square), and that the location of other objects is known (other grid squares). The robot's actions simply move the robot from its current square to a neighbouring square. Unpredictability of movement is sometimes modelled by occasionally moving the robot in a different direction. However, once the robot has moved its location is assumed to be known.

Grid world tasks are usually Markovian environments, in which the probability distribution of the next-state is affected only by the current state and action. For robotics this is inappropriate: it is impossible to measure the state of the environment and robot with sufficient completeness and accuracy to produce a Markovian environment.



(a) The 'robot' (□) must learn to reach a location (★) while avoiding fixed 'obstacles' (■)



(b) The robot must learn to reach a location by controlling its actuators in response to information from sensors

Figure 2.2: There are noticeable differences between a typical grid world task (a), and a typical real environment task (b).

Most reinforcement learning algorithms and simulations deal with discrete representations. However, many robot sensors report continuously variable results, such as positions and forces, and most robot actuators are capable of continuously adjusting their output. Discretising the continuous world so that discrete reinforcement learning can be applied creates problems, including information loss and coarseness of control. The inability to completely measure or represent the state is known as the *perceptual aliasing* problem (Whitehead and Ballard, 1992). When perceptual aliasing occurs, differing environment situations are represented as the same state. The learning system can be misled since it only has access to its state representation, not to the real situation.

Despite the deficiencies described above, simulations can sometimes generate valuable data. Controllers learnt in simulation have controlled robots successfully, after further learning on the robot (Asada et al., 1994; Benbrahim and Franklin, 1997). In these cases the researchers were already experienced in robotics and had a strong understanding of the problems; they were able to refine their simulations and learning systems iteratively by switching between using the robot and using the simulation.

Another example of successful transfer from simulation to a real robot is the helicopter controller developed by Bagnell and Schneider (2001). The controller performed hovering and trajectory following, controlling the helicopter's pitch, roll, and horizontal translation. The task of the learning system was to find 10 control parameters; the structure of the controller was pre-configured into two, 5 parameter controllers. The stability and robustness of the controller were enhanced by allowing for a non-Markovian environment (Ng and Jordan, 2000). Two existing helicopter models were used to validate the controller before evaluation on the robot. Although the controller was developed in simulation, the simulation model was developed based on data recorded on the helicopter.

To interpret problems attempted only in simulation is misleading since most of the real robotics problems are abstracted away. In this thesis, experiments performed purely in simulation are not regarded as 'robot' experiments. Algorithms for robots must be validated on real robots.

Considering the problem of slow learning, the unsuitability of simulation, and the lack of spectacular past achievements, one could wonder why reinforcement learning systems have not already been abandoned as a tool for robotics. Nevertheless, the desire exists for robots to exhibit novel behaviour; it is not possible to pre-program suitable behaviour for all situations.

Adaptive control systems are designed to cope with uncertainty in parameters; however, this is not sufficient to produce novel behaviour. A learning system can produce novel behaviour by solving *constructional* problems. In particular, problems that require coordinated, predictive actions over multiple time steps. Although this thesis focuses on *model-free* reinforcement learning systems, this is not the only approach for learning to solve multiple step control problems.

2.2.2 Model-Based Learning systems

Many methods for solving multiple step control problems use a dynamic model, either during a learning process, or during control execution (Bitmead et al., 1990; Narendra and Parthasarthy, 1994; Hunt et al., 1994; Werbos, 1992). A dynamic model describes the next-state as a function of the current state and actions. Modelling is related to simulation; however modelling usually refers to lower level interaction between sensors and actuators. Additionally, models are usually structured so as to allow mathematical manipulation in the design of a controller (Bitmead et al., 1990).

The source of the model could be measurement and mathematical modelling of the system, often requiring calibration. A more adaptable method is to define a model with some flexibility in its parameters to account for unknown characteristics. These parameters can then be found through experimentation. In control theory this is known as system identification. A still broader approach is to make minimal assumptions about the system and learn the entire dynamic model through supervised learning.

The Backpropagation Through Time (BTT) algorithm is an example of a model-based method for learning to solve multiple step control problems (Nguyen and Widrow, 1990). In BTT errors in control are propagated backwards over multiple time steps through a learned dynamic model. The model is used open-loop to calculate the control policy, without interaction with the real system. BTT's heavy reliance on the accuracy of its learned dynamic model makes it sensitive to modelling errors (Werbos, 1992).

Identifying or learning a model then finding a controller for that model can lead to problems since the model may not describe important characteristics of the system. An inexact model has finite approximation resources. If these resources are used in modelling the system at an unimportant frequency range, unimportant regions of the state space, or for unimportant state variables, the controller design suffers (Bitmead et al., 1990). The controller could be perfectly adapted to controlling the model while being useless for controlling the real system.

Adaptive Model Predictive Control (MPC) using an artificial neural network is another model-based method for learning to solve multiple step control problems (see Hunt et al., 1994). At each time step, this method searches for the best control action by predicting the state of the system over several time steps using a learned dynamic model. MPC interacts with the real system more than the BTT algorithm; it is more closed-loop. Consequently, it is less sensitive to modelling errors. However, neural network-based MPC introduces a difficult, time-consuming optimisation problem every time an action is to be executed.

Although reinforcement learning is generally regarded as model-free, an available model can be exploited. In Sutton's (1990) Dyna system a learned model is used to generate additional learning experiences for an otherwise model-free reinforcement learning system. Werbos's (1992) adaptive critic family of methods use a learned dynamic model to assist in assigning reward to components of the action vector. Thrun (1995) used the framework of explanation-based neural networks to refine action-values through a dynamic model. An important contribution of Thrun's work is that it discussed the importance of estimating the accuracy of the model over the state space and weighting the influence of the model based on the estimate. Without this step an inaccurate model could misdirect the reinforcement learning system.

Whether it is more efficient to learn a control policy directly, or to learn a model first and then find an appropriate policy, or to simultaneously learn a model and policy, depends on the particular learning problem.

In well-defined, control-like problems, in which a model of the system is available with only a few uncertain parameters, model-based adaptive or robust control systems may be the most appropriate solution. Additionally, interaction between a controller in the process of learning and the real system is sometimes not feasible. For example, when controlling a chemical processing plant in a factory it should be possible to construct a dynamic model, based on the theoretical and measured characteristics of the process. Interaction between an unpredictable controller and the plant is likely to be dangerous. In that situation, a system that learns, from scratch, through interaction with the plant would be a poor selection.

Model-free learning systems may be more applicable to constructional problems for which there is no existing model. Furthermore, in constructional problems there may be no clear path towards developing a dynamic model (Baker and Farrel, 1992). For example, developing a dynamic model or simulation suitable for learning to navigate a robot from room-to-room, avoiding obstacles using vision, seems much more difficult than learning to solve the problem directly, in the real environment.

The following section introduces a model-free reinforcement learning algorithm, $\mathcal{Q}$-learning, through describing model-based dynamic programming.

2.3 Dynamic Programming and $\mathcal{Q}$-Learning

Dynamic programming is a model-based approach to solving reinforcement learning problems (Bellman, 1957). One form of dynamic programming stores the expected value of each action in each state. The *action-value*[1], $\mathcal{Q}$, of an action in a state is the sum of the reward for taking that action in that state plus the expected future reward if the policy described by the stored action-values is followed from then on:

$$\begin{aligned}\mathcal{Q}(x_t, u_t) = E\Big[& r_t(x_t, u_t, X_{t+1}) \\ & + \gamma r_{t+1}\left(X_{t+1}, \arg\max_{u_{t+1}} \mathcal{Q}(X_{t+1}, u_{t+1}), X_{t+2}\right) \\ & + \gamma^2 r_{t+2}\left(X_{t+2}, \arg\max_{u_{t+2}} \mathcal{Q}(X_{t+2}, u_{t+2}), X_{t+3}\right) + \ldots\Big]\end{aligned} \tag{2.2}$$

[1]We will generally refer to the value of an action in a state as the action-value, and the collection of values of actions as the action-values; rather than using the terms $\mathcal{Q}$-values, $\mathcal{Q}$-table, and $\mathcal{Q}$-function. The convention is based on suggestions by Sutton (2001).

where random (probabilistic) variables are denoted by capital letters. $\arg\max_u \mathcal{Q}(x,u)$ is the action with the highest value in state x. $\max_u \mathcal{Q}(x,u)$ is the value of the highest valued action in state x—this is called the *value of state* x. The action values are denoted $\mathcal{Q}^*$ if they satisfy *the Bellman optimality equation*:

$$\mathcal{Q}^*(x_t,u_t) = E\left[r_t(x_t,u_t,X_{t+1}) + \gamma \max_{u_{t+1}} \mathcal{Q}^*(X_{t+1},u_{t+1})\right] \quad \forall x_t,u_t \tag{2.3}$$

Performing $\arg\max_u \mathcal{Q}^*(x,u)$. . . is guaranteed to be an optimal policy. In this framework, the problem of finding an optimal policy is transformed into searching for $\mathcal{Q}^*$. The dynamic programming approach finds $\mathcal{Q}^*$ iteratively through a forward dynamic model.

$\mathcal{Q}$-Learning is a model-free approach for finding $\mathcal{Q}^*$ (Watkins, 1989; Watkins and Dayan, 1992). In $\mathcal{Q}$-learning real world experience takes the place of the dynamic model—the expected values of actions in states are updated as actions are executed and the effects can be measured. In one-step $\mathcal{Q}$-learning action-values are updated by the one-step $\mathcal{Q}$-update equation:

$$\mathcal{Q}(x_t,u_t) \xleftarrow{\alpha} r(x_t,u_t,x_{t+1}) + \gamma \max_{u_{t+1}} \mathcal{Q}(x_{t+1},u_{t+1}) \tag{2.4}$$

where α is a learning rate (or step size) between 0 and 1 that controls convergence. The arrow in Equation (2.4) is the *move-toward* operator, it should not be confused with logical implication. The operation $A \xleftarrow{\alpha} B$ is equivalent to moving A toward B in proportion to α. A and B are scalars or vectors. If α is not shown it is equivalent to 1.

$$A \xleftarrow{\alpha} B \text{ , is equivalent to,}$$
$$A := (1-\alpha)A + \alpha B, \qquad \alpha \in [0,1]$$

Under the one-step $\mathcal{Q}$-learning update the action-values are guaranteed with probability 1 to converge to optimal action-values ($\mathcal{Q}^*$) under the following conditions (Watkins and Dayan, 1992):

1. each action is executed in each state an infinite number of times;
2. α is decreased with an appropriate schedule; and
3. action-values are stored perfectly (as in a table).

True convergence to optimal action-values is rarely achievable. In practical use the goal is that the action-values will describe an acceptable controller in reasonable time.

Q-Learning represents the values of all actions in all states, rather than only representing the policy. The knowledge is expensive in terms of the amount of information that has to be stored; however, it increases the learning system's flexibility by supporting *off-policy learning*.

2.4 Off-Policy Learning

A *policy* is a definition of behaviour: a mapping from states to actions. The policy that is being followed is the *behaviour* policy. In Q-learning the *estimation* policy is to perform the action with the highest estimated action-value. The Q-learning updates refine the action-values to bring the estimation policy closer to an optimal policy. In one-step Q-learning the *behaviour* policy can be unrelated to the *estimation* policy (Sutton and Barto, 1998). The property is known as *off-policy* learning. Conversely, in *on-policy* learning the behaviour policy is the same as the estimation policy. Q-Learning's off-policy learning capability allows it to learn an optimal policy whilst following any behaviour policy that has a non-zero probability of selecting all actions. A simple test for off-policy learning capabilities is to perform the learning update based on a large database of random of actions in states; an off-policy learner should be able to learn an acceptable policy from the information.

There is a spectrum of variations around the off-policy learning idea. The main theme is that the behaviour policy can be unrelated to the estimation policy. Two dimensions of variation can be examined. The first variation is whether the behaviour policy is *known* or *unknown*. Methods capable of learning when the behavioural policy is unknown can also learn when the policy is known.

The second variation is whether the learning system learns from a *sequence* of states, actions and rewards through time, or if it learns from *non-sequential* examples of state, action, next-state, and reward. Following the non-sequential definition of off-policy learning, to some researchers 'off-policy' implies 'without following any policy', or any sequence of actions. Some researchers use the term 'simulation' for sequential situations and 'model' for non-sequential situations: users of a 'simulation' cannot set the system to an arbitrary state, while 'models' allow immediate access to any state. Using this terminology an on-line experiment on a real robot would be classed as a 'simulation'. This is clearly unacceptable in robotics research; therefore, the terms *sequential* and *non-sequential* are used in this thesis.

Table 2.1 classifies various model-free reinforcement learning methods. Algorithms that can only learn on-policy include Actor-Critic (Barto et al., 1983), Sarsa (Rummery and Niranjan, 1994) and direct gradient ascent approaches (Gullapalli, 1990; Williams, 1992; Kimura and Kobayashi, 1998; Sutton et al., 1999a; Baxter et al., 1999). These systems allow occasional non-estimation policy actions in order to explore and improve the policy; however, they cannot learn from an extended sequence of non-estimation policy actions.

Table 2.1: Categorisation of off-policy and on-policy learning algorithms

off-policy	**unknown policy**	**known policy**
non-sequential	one-step $\mathcal{Q}$-learning (Watkins and Dayan, 1992)	
sequential	$\mathcal{Q}(\lambda)$ with tree backup (Precup et al., 2000)	$\mathcal{Q}(\lambda)$ with per-decision importance sampling (Precup et al., 2000)

becomes one-step $\mathcal{Q}$-learning	$\mathcal{Q}(\lambda)$ (Watkins, 1989), C-TRACE (Pendrith and Ryan, 1996)

on-policy only	Actor-Critic (Barto et al., 1983), Sarsa (Rummery and Niranjan, 1994), direct gradient ascent (Gullapalli, 1990; Williams, 1992), $\mathcal{Q}(\lambda)$ (Peng and Williams, 1996)

One-step $\mathcal{Q}$-learning is a non-sequential learning method. Since each update passes rewards only one step backwards through time one-step $\mathcal{Q}$-learning cannot exploit the information present in sequences of experiences. Rather than learn directly from the sequence of rewards, one-step $\mathcal{Q}$-learning depends on bootstrapping, relying heavily on the accuracy of its internally represented estimates of value. Use of bootstrapping increases learning performance; however, over-reliance on bootstrapping causes instability (Sutton and Barto, 1998). Sequential learning methods that rely less heavily on bootstrapping can be expected to be more stable, and learn from experience more efficiently, than one-step $\mathcal{Q}$-learning.

Historically, sequential learning algorithms have not combined well with off-policy learning. TD(λ) is a well-known sequential learning algorithm (Sutton, 1988; Sutton and Barto, 1998). The λ parameter controls the mix between bootstrapping and measuring rewards over time. Thus, reward is passed back over more than one step when $\lambda > 0$. One-step $\mathcal{Q}$-learning is equivalent to TD(0). There are several '$\mathcal{Q}(\lambda)$' algorithms that combine $\mathcal{Q}$-learning and TD(λ). The degree of off-policy learning capabilities of the $\mathcal{Q}(\lambda)$ algorithms has been mixed. Peng and Williams's (1996) method assumes that the number of non-estimation policy actions is small. Watkins's (1989) $\mathcal{Q}(\lambda)$ method requires that λ be set to zero for non-estimation policy actions (Sutton and Barto, 1998, p. 182). Similarly, Pendrith and Ryan's (1996) C-TRACE algorithm measures rewards over time until a non-estimation policy action is executed, in which case the one-step $\mathcal{Q}$-learning update is used. Watkins's $\mathcal{Q}(\lambda)$ and the C-TRACE algorithm both become one-step $\mathcal{Q}$-learning if all actions are non-estimation policy actions. Accordingly, they are just as capable of off-policy learning as one-step $\mathcal{Q}$-learning, and have additional sequential learning capabilities when following the estimation policy. However, these algorithms require the detection of non-estimation policy actions. When actions are continuously variable, rather than discrete, it is questionable which actions follow the estimation policy, and therefore, whether the switch should be made between using actual rewards or estimated value.

Recently the statistical framework of importance sampling has led to a variation of $\mathcal{Q}(\lambda)$ that avoids the need to decide if an action follows the estimation-policy (Precup et al., 2000). The tree backup algorithm multiplies λ by a factor that represents the probability that the estimation policy would have chosen that action. The probability replaces the problematic decision of whether an action exactly follows the estimation policy; consequently, the approach is applicable for continuous action problems.

2.4.1 Off-Policy Learning Techniques

Off-policy learning techniques can reduce the time that a learning robot is under the control of a poorly performing, unpredictable controller. This time period is more important for robotics than the total learning time or the off-line processing time. Off-policy learning techniques are especially suitable for systems that include existing controllers, other behaviours or existing knowledge, in addition to the learning system; rather than the classical reinforcement learning situation in which the learning system learns from scratch, interacting purely with its environment. The following off-policy learning techniques are discussed:

- flexible exploration;
- experience replay;
- learning from other controllers;
- learning from other behaviours;
- learning in hierarchical systems;
- learning from phantom targets; and
- learning through data transformations.

Flexible exploration

Since off-policy learning algorithms increase the range of possible exploration methods, they are termed *exploration insensitive* (Kaelbling et al., 1996). Exploration is the practice of trying non-policy actions in an attempt to improve the policy. On-policy methods explore by making slight variations around the current policy. More efficient algorithms can be used when off-policy learning is available. For example, the learning system can try to reach regions of state space that have been explored less frequently (Thrun, 1992; Wilson, 1996). This capability of off-policy learning systems is well-known.

Experience replay

A learning *experience* can be defined as an ordered set of state, action, next-state, and reward. If learning experiences from previous experiments or the current experiment are replayed the learning system can learn from these experiences as if they are being re-experienced (Lin, 1992).

The main motivation for replaying experience data is that new data can be expensive. In simulation all data is cheap. However, in the real world measured data is the scarcest resource; it is more expensive than memory or processing time. The purpose of replaying experiences is to extract as much knowledge as possible from existing data. Off-policy learning is necessary for experience replay since the actions taken in the replayed experiences need not follow the *current* estimation policy.

Obviously, replaying experiences assumes that the learning system's environment has not changed since the data was stored. If circumstances have changed only slightly, for example if a camera has been bumped, learning by replaying data could still be a good first step to learning under the new circumstances.

Learning from other controllers

Quite often existing controllers or demonstrators are available that are more or less capable of carrying out the task that is to be learnt. Using off-policy learning a learning system can passively assess and learn from the actions of existing controllers. It is different from the common supervised learning or behavioural cloning system in which the learning controller simply learns to emulate the existing controller.

An example of the learning from other controllers approach would be if four people independently designed gait controllers for a walking robot. The reinforcement learning system could learn from the actions of these four controllers to develop its own hybrid.

The learning controller could then continue to refine its behaviour on-line.

Several researchers have demonstrated reinforcement learning systems that can learn from a teacher (Lin, 1992; Saito and Fukuda, 1994; Clouse, 1996; Zhang, 1996; Ryan and Reid, 2000; Smart and Kaelbling, 2000). Learning from a teacher is learning from one other controller. In Smart and Kaelbling's work the task was to control a robot moving down a corridor. Learning time was decreased by allowing the system to observe the actions of a hand-coded controller.

Learning from other behaviours

The *learning from other controllers* idea can be extended to the idea of *learning from other behaviours*. Behavioural systems separate the overall robotic control problem into simpler problems handled by behaviours. The subsumption architecture is an example of a behaviour-based approach (Brooks and Mataric, 1993). The behaviours operate in parallel, each handling different aspects of the task.

Learning from other behaviours could be applied, for example, to a behaviour-based target seeking robot. The target seeking behaviour is dominant by default. It is accompanied by a contour following behaviour to help the robot find its way around obstacles. When an obstacle is detected, the contour following behaviour becomes dominant and controls the robot's actions. In the same way that an off-policy learning system can passively assess the actions of another controller, it can also assess the actions of a different behaviour. In this case the target seeking behaviour could learn from the actions of the contour following behaviour. Even if most of the actions of the contour following behaviour are inappropriate for the task of target seeking, assessing these actions can make a contribution to learning the target seeking behaviour by helping to eliminate unprofitable areas of the state action space.

Learning in hierarchical systems

Hierarchical reinforcement learning architectures may allow the reinforcement learning approach to scale to larger tasks. The approach aims to simplify the learning system's task by adding more structure to the problem and exploiting redundancies. Off-policy learning has been suggested as an important tool for hierarchical reinforcement learning (Dietterich, 2000; Sutton et al., 1999b; Kaelbling, 1993a). In these systems off-policy learning allows experiences to be evaluated as part of many sub-policies in parallel.

Ryan and Reid's (2000) simulated aeroplane controller used a hierarchical Q-learning system with off-policy learning. The *gentle ascend* sub-policy, for example, could learn from actions generated by the *straight and level* sub-policy which was in the next level of the hierarchy. However, the experiences gathered were only used for the training of currently activated parts of the sub-policy hierarchy, rather than all sub-policies.

Learning from phantom targets

Reinforcement learning problems are often formulated in terms of achieving a goal. In many systems multiple goals are possible. Separate behaviours can be used to achieve *qualitatively* different goals as described in the previous section. *Quantitatively* different targets, for example different positions or velocities, can be achieved with a single behaviour. In these problems the state representation is a function of both the current state and the target state.

As an example, consider a dart throwing robot. The current target is a specific, numbered section on the dart board. The state could be composed of parameters describing the robot position and pose combined with the current target. When the robot throws the dart the learning system receives a reward of 1 for hitting the current target, or -1 for missing. Since the current target is part of the state representation the dart throwing experience can be used to update the value functions for all of the other targets on the dart board. These other targets can be referred to as *phantom targets*. The reward used to update the value function for the phantom targets is different from the reward for the current target.

Kaelbling (1993b) uses the term *all goals updating* for this learning idea. The term has not been used in this thesis as the term has also been applied to describe other off-policy learning techniques that we have classified differently.

Learning through data transformations

It is sometimes possible to make better use of information by exploiting physical characteristics of the system through data transformations. For example, if there is a known physical relationship between the left leg and the right leg of a robot every experience can be mirrored to produce another valid experience. Actions performed in the transformed data do not follow the estimation policy. An alternative to data transformation is to represent states and actions in a way that explicitly encodes the relationship.

2.4.2 Practical Off-Policy Learning

The major motivation for the use of one step Q-learning in this thesis is the algorithm's ability to learn off-policy. Off-policy learning supports state and action space exploration and it allows learning from experience replay, other controllers, other behaviours, phantom targets, and data transformations. Off-policy learning has particular application to behavioural and hierarchical systems.

It could be argued that some of the off-policy learning techniques that have been described are the same technique. For example, *learning from other controllers* could be said to be the same as *learning from other behaviours*; and *learning in hierarchical systems* could be said to be the same as *learning from other behaviours* or *learning from other goals*. The differences are more conceptual than actual. In this research work they are separated in order to stimulate ideas and assist description.

In a practical system the techniques may blend and combine together. Data could be gathered from experiments using existing controllers, some implementing different behaviours than the desired behaviour of the learning system. The stored data could be repeatedly reprocessed by the learning system and expanded through the use of phantom goals and data transformations. This could be considered as a *passive learning* stage. The controller could then be further refined through interaction with the real world in an *active learning* stage. The active learning stage could include off-policy exploration methods to reach regions of the state-action space that were not adequately explored during the passive learning stage. Smart and Kaelbling's (2000) research demonstrated a range of off-policy learning techniques, including passive and active learning stages, for a robot corridor following task. They found that the controller learnt through passive learning had higher performance than the hand-coded controller used to generate the experiences.

This discussion of off-policy learning methods has assumed a table-based representation of the action-values in which they can be stored exactly. Chapter 4 discusses some of the issues that arise when the action-values are stored inexactly, by an approximator. Approximate storage is useful in systems that operate in large or continuous state spaces.

2.5 $\mathcal{Q}$-Learning with Continuous States and Actions

Many real world control problems, particularly in robotics, require actions of a continuous nature, in response to continuous state measurements. Therefore, a robot control system should be capable of producing actions that vary *smoothly* in response to smooth changes in state. This section discusses continuous state and action $\mathcal{Q}$-learning systems. The emphasis is on $\mathcal{Q}$-learning systems since they support off-policy learning.

2.5.1 Continuous States

The real world is continuously valued; positions, forces, and velocities rarely have a discrete range of possible values. Robotics problems often involve mapping, accurate positioning, manipulation, or trajectory tracking.

As a pedagogic example, consider an archery robot. The robot's task is to learn to hit the centre of the target shown in Figure 2.3. The robot aims its arrows using a misaligned sight. The robot learns through reinforcement learning. Reward is highest at the centre of the target. The state is based on the previous arrow: the sighting when the arrow was shot; and where the arrow hit the target. The action is the next choice of sighting. To

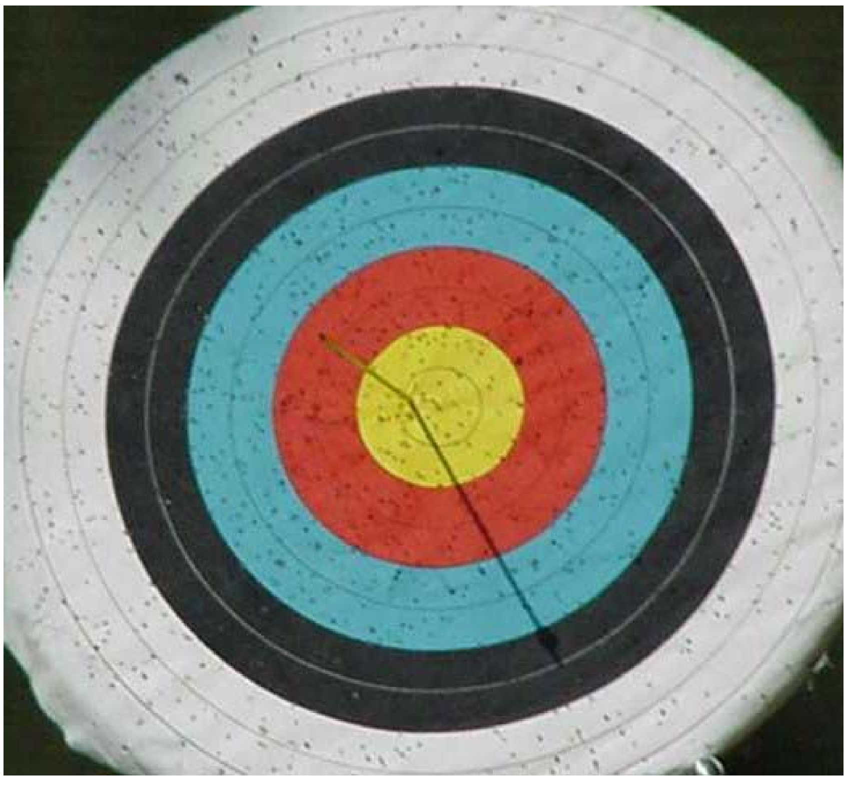


Figure 2.3: Archery target

initialise the system the robot first fires an arrow under the assumption that the sight is perfectly aligned. After the first shot the robot repeatedly updates its aim based on the results of previous shots.

In order to solve the problem within the framework of Q-learning state and action representations must be designed and the action-values must be stored. In the classical implementation of Q-learning the action-values are stored in a look-up table; continuous variables are discretised to become an index into the table. Discretising the state of the world is a common approach in artificial intelligence research. It introduces a number of problems. If the state is coarsely discretised a *perceptual aliasing* problem occurs.

Since many different world states will be mapped to a single represented state information will be lost. A coarsely discretised view of the archery target is shown in Figure 2.4. Because of the discretisation of the state the robot has only low accuracy information about the destination of the last arrow or the last sighting position. The discretisation in the figure is finer than is usual for reinforcement learning. Often continuous state variables are discretised to only two or three levels: near, far; left, centre, and right; fast, slow, and stopped; or leg up and leg down.

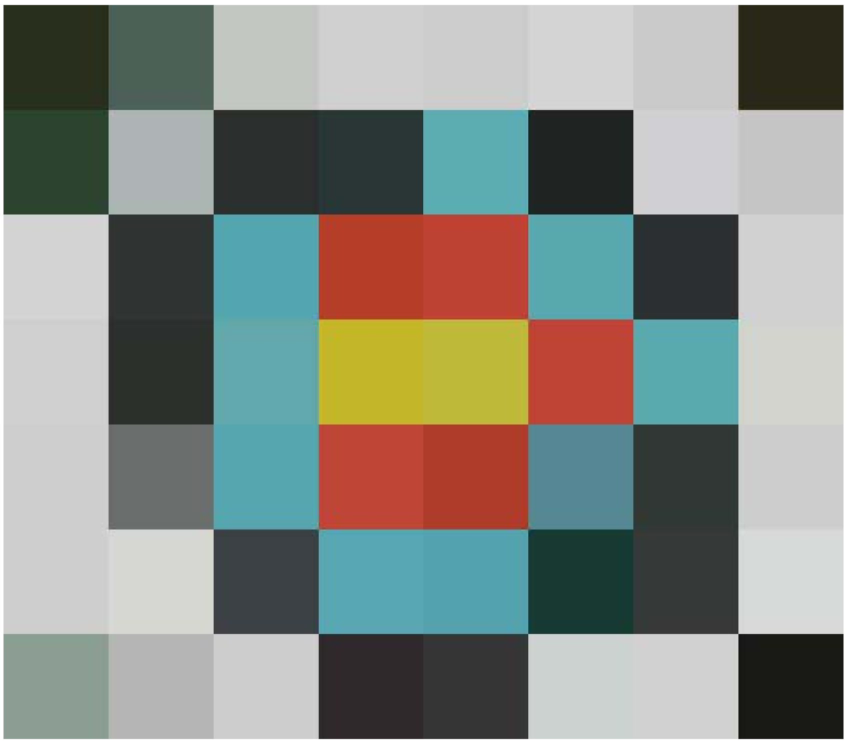


Figure 2.4: The archery target coarsely discretised by position

The perceptual aliasing problem can be mitigated by discretising finely. Figure 2.5 shows the archery target finely discretised. The robotic archer should now be able to correct its aim based on the more accurate state information. A disadvantage is that discretising finely increases the amount of memory required to store the action-values; memory usage also increases exponentially with the number of state variables. This is the curse of dimensionality (Bellman, 1957).

A serious side effect of discretisation is that classical $\mathcal{Q}$-learning does not generalise between cells in the table. Relationships, such as near, far, above, and below, are not preserved. In effect the representation of the archery target is as in Figure 2.6. The robot archer must learn about each cell in its look-up table independently. As the fineness of the discretisation and the number of state variables increases, so does the number of experiences required to fill the table. Further, $\mathcal{Q}$-Learning requires repeated visits to each state. Under fine discretisation the learning time becomes intractable. If discretisation is used a compromise is made between introducing perceptual aliasing and speed of learning. Fine discretisation fails due to the inability to generalise: similar states should be regarded as having similar properties unless there is evidence to the contrary. Generalisation has long been recognised as an important ingredient for practical reinforcement learning (Sutton and Barto, 1998, pp. 224–226).

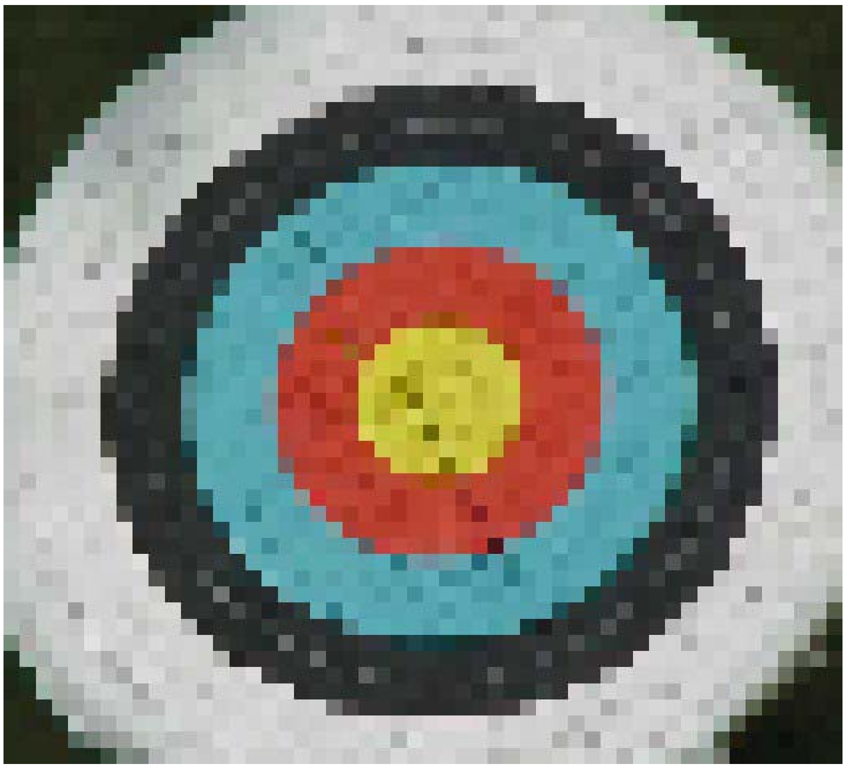


Figure 2.5: Finely discretised archery target

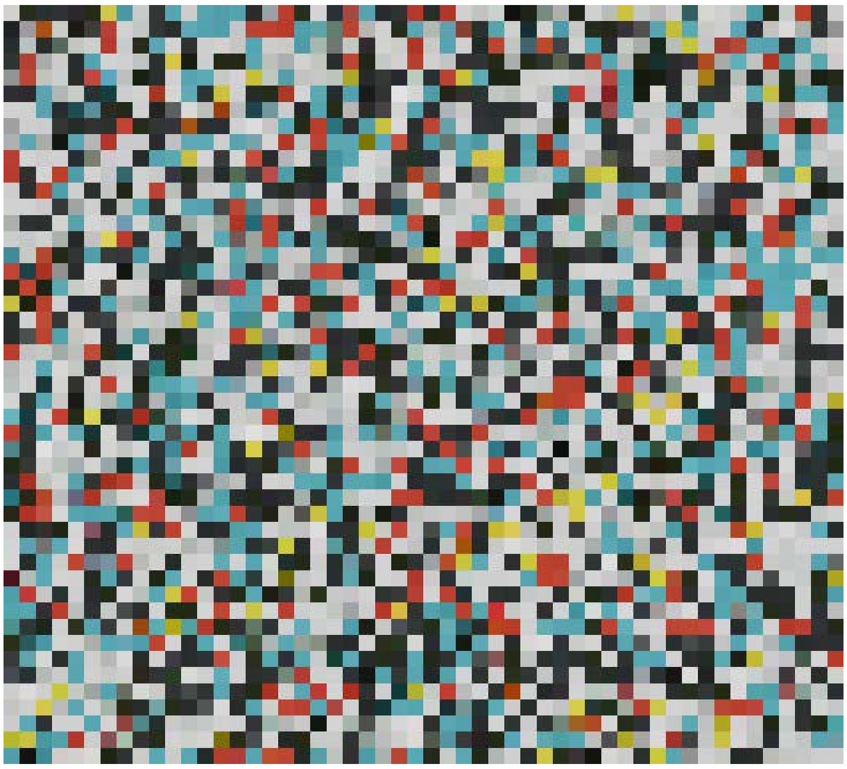


Figure 2.6: Finely discretised archery target without generalisation by position

Generalisation can be introduced by abandoning table-based storage of the action-values in favour of a function approximator. Figure 2.7 shows the archery target as seen through a function approximator. Although detail is lost, relationships, such as near, far, above, and below, are generally preserved. Small changes in state variables are represented. Lin's (1992) QCON system is an example of the function approximation approach. The system used a number of artificial neural networks. Neural networks can be trained incrementally, through backpropagation of errors, to represent a non-linear mapping between inputs and outputs (Rumelhart et al., 1986). The QCON system used one artificial feedforward neural network for each possible action. The input to the neural networks was the continuous state; the output was the value of that neural network's action. The neural networks were trained to output the correct action-values.

Like most continuous state systems based on function approximation, the QCON system still represents actions discretely. Figure 2.8 illustrates the problem of coarsely discretised actions. The robot can only sight a few predefined locations; its chances of shooting the middle of the target are small. The following section discusses Q-learning with continuously variable actions.

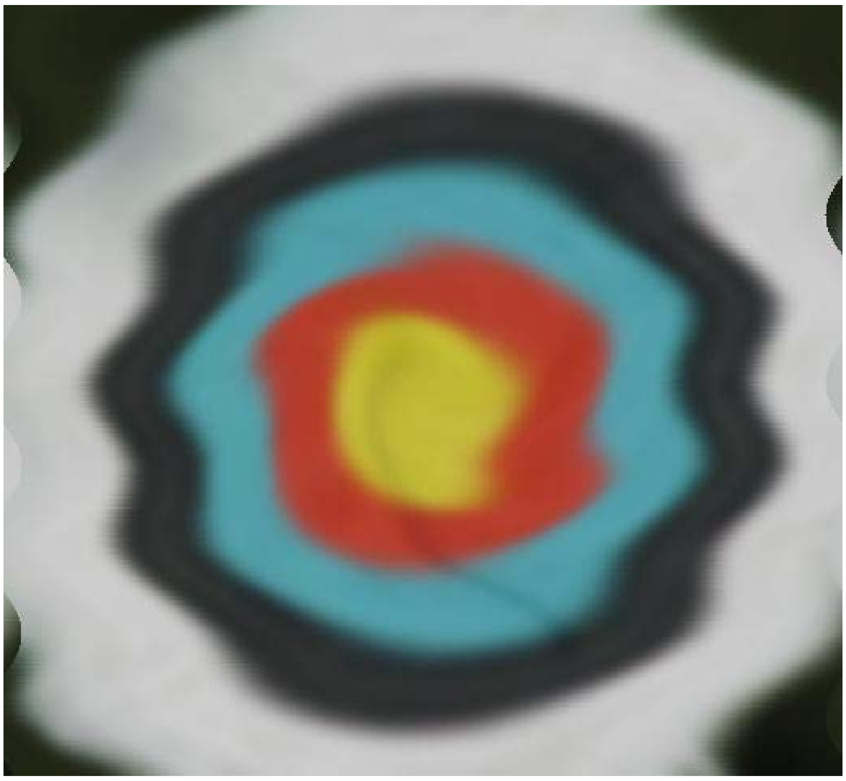


Figure 2.7: Archery target represented through function approximation

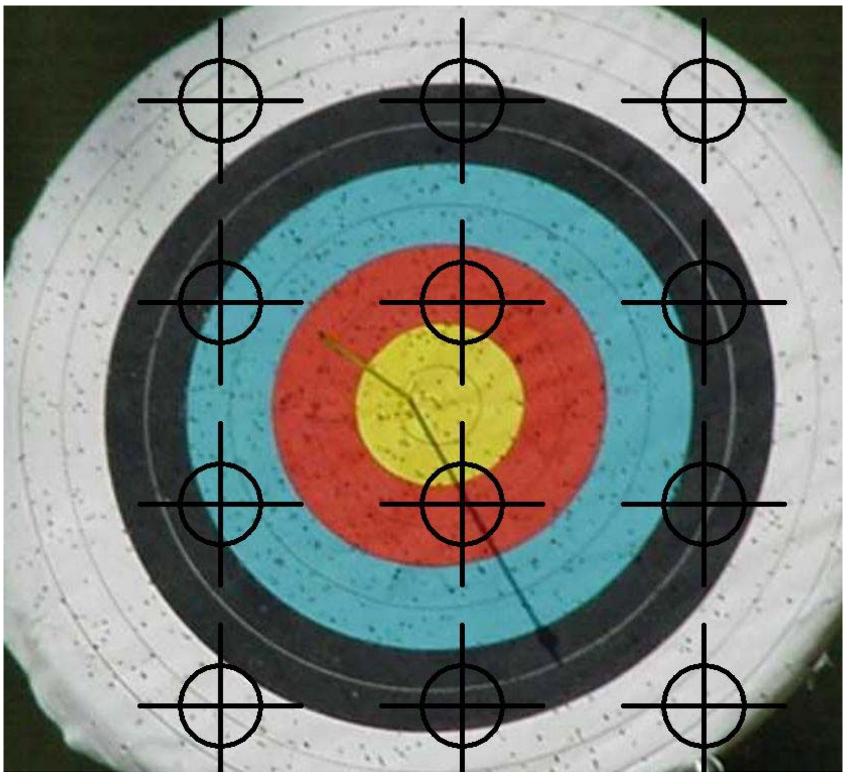


Figure 2.8: Coarsely discretised actions (sightings)

2.5.2 Continuous Actions

Interaction with the real world requires smoothly variable actions as well as continuous state representations. Learning systems that discretise actions suffer in similar ways to those that discretise states—high memory use and slow learning due to inability to generalise. In addition it is often desirable that control actions vary smoothly with the state. Most robot actuators are capable of continuously adjusting their output. Continuous actions give more accurate and predictable movement, reduce wear, and improve energy consumption. Figure 2.9 illustrates the concept of continuously variable actions. The robotic archer can sight any point on the target.

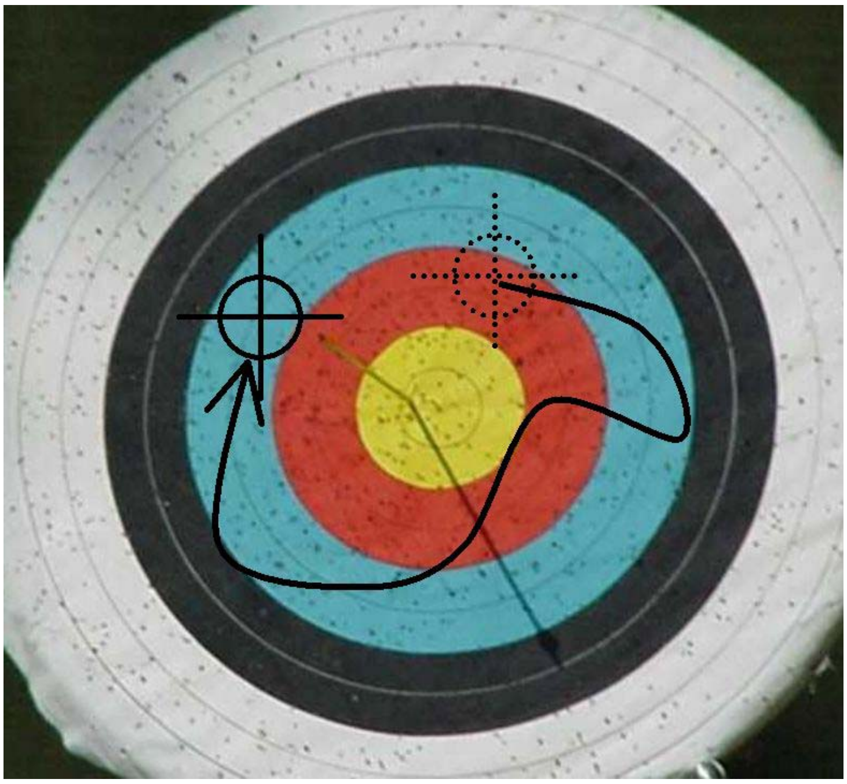


Figure 2.9: Continuously variable actions (sightings)

The following sections define necessary properties for continuous state and action $\mathcal{Q}$-learning and survey existing methods.

2.5.3 Necessary Properties

The focus of this thesis is practical continuous state and action $\mathcal{Q}$-learning for robotics. An appropriate algorithm will handle continuously variable states and actions as well as the basic requirements for $\mathcal{Q}$-learning. In addition, the algorithm must satisfy real-time constraints. In a real-time system the time at which an event occurs is as important as what occurs. All robots are real-time systems: they must process sensor data and react in a timely manner. Real-time behaviour requires speed, predictability, and consistency. The situation is unlike that of simulation, where the only time requirement is that the result is ready before the money runs out. Another requirement for robotics is fast learning. It is not possible to assess learning speed by inspecting the learning architecture, and simulations can be misleading. An indicator of learning speed is the ability to learn tasks on a real robot.

This research work proposes that there are eight criteria that are necessary and sufficient for a system to be capable of truly continuous state and action $\mathcal{Q}$-learning in real-time (refer to Figure 2.10). These requirements could be considered pedantic; incompletely satisfying one of these conditions does not guarantee that an algorithm will not perform adequately in practice in a particular application.

Action Selection:	Finds the *action* with the highest expected value in real-time.
State Evaluation:	Finds the *value* of a *state* as required for the $\mathcal{Q}$-update, Equation (2.4).
$\mathcal{Q}$ Evaluation:	Stores or approximates all action-values as required for the $\mathcal{Q}$-update, Equation (2.4).
Model-Free:	Requires no model of system dynamics to be known or learnt.
Flexible Policy:	Allows representation of a broad range of policies to allow freedom in developing a novel controller.
Continuity:	Actions can vary smoothly with smooth changes in state.
State Generalisation:	Generalises between similar states, reducing the amount of exploration required in state space.
Action Generalisation:	Generalises between similar actions, reducing the amount of exploration required in action space.

Figure 2.10: Essential capabilities for a continuous state and action $\mathcal{Q}$-learning system

Criterion *Action Selection* is required when choosing an action to execute. It allows extraction of the *policy*. A system that fulfils this criterion can calculate $\arg\max_u \mathcal{Q}(x, u)$ under real-time constraints. Any control system must select actions in real-time if it is to act in a dynamic environment. There is also a perceptual aliasing problem if there is an appreciable delay between estimation of the state and execution of the action. It occurs because the action was not executed in the state as recorded: the state changed while the action was being selected.

Criterion *State Evaluation* is required during the update of the action-values. The system must be able to calculate the value of any state. A state's value is the value of highest valued action in that state: $\max_u \mathcal{Q}(x, u)$. Criterion $\mathcal{Q}$ *Evaluation* allows off-policy learning by representing the values of all actions in all states. The system must represent $\mathcal{Q}(x, u)$ for arbitrary x and u.

The *Model-Free* criterion reflects the ability of $\mathcal{Q}$-learning systems to learn without access to a dynamic model of the system to be controlled. Model-based systems are useful and practical; however, they address a different class of tasks, as discussed earlier.

Criterion *Flexible Policy* omits systems that can only describe a overly restricted set of controllers—a controller whose actions must be a linear function of the state would not pass this criterion. If design of the controller only requires tuning of a few parameters then the problem is adaptive control, not learning.

Criterion *Continuity* requires policies in which actions can vary smoothly with smooth changes in state (see Figure 2.9). It does not imply that actions *must* always vary smoothly in response to smooth changes in state. A good policy sometimes blends smooth changes with sudden changes (Takeda et al., 2000). An example is a controller that moves a camera smoothly while pursuing a moving target, then moves suddenly to move a new target into view.

Many of the systems described in the following section cannot fulfil criterion *Continuity*. In these systems the function that maps state to action is a staircase—a piecewise constant function. Although the height and width of the step may be continuously variable, a smooth change in action is impossible. Technically, a constant output could be considered to be continuous (i.e. without discontinuities), but criterion *Continuity* in this thesis means that the output can *vary* smoothly. Figure 2.11 contrasts a continuous function with a piecewise constant function. If the robot archer learnt piecewise constant actions it would have only a fixed set of possible actions (Figure 2.12). The fixed set of actions would be learnt, but cannot vary with the state; thus smooth changes in action are impossible.

Without the ability to generalise (criteria *State Generalisation* and *Action Generalisation*) learning in continuous environments is impossible. A real valued state or action cannot repeat itself exactly; hence without generalisation gathered experiences are useless. Figure 2.6 is Figure 2.5 modified to remove generalisation between positions.

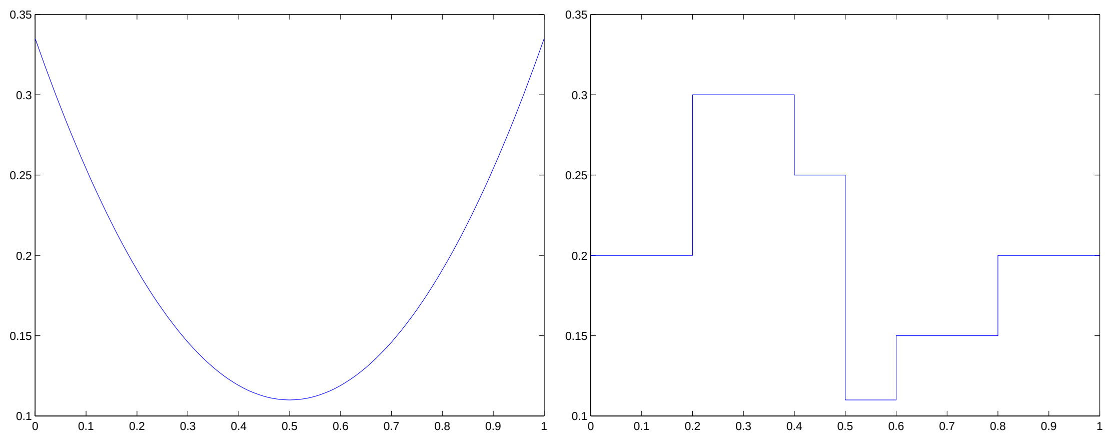


Figure 2.11: In a continuous function (*left*) or a piecewise continuous function the output can vary smoothly with the input. In a piecewise constant function (*right*) the output is like a staircase and never varies smoothly with the input.

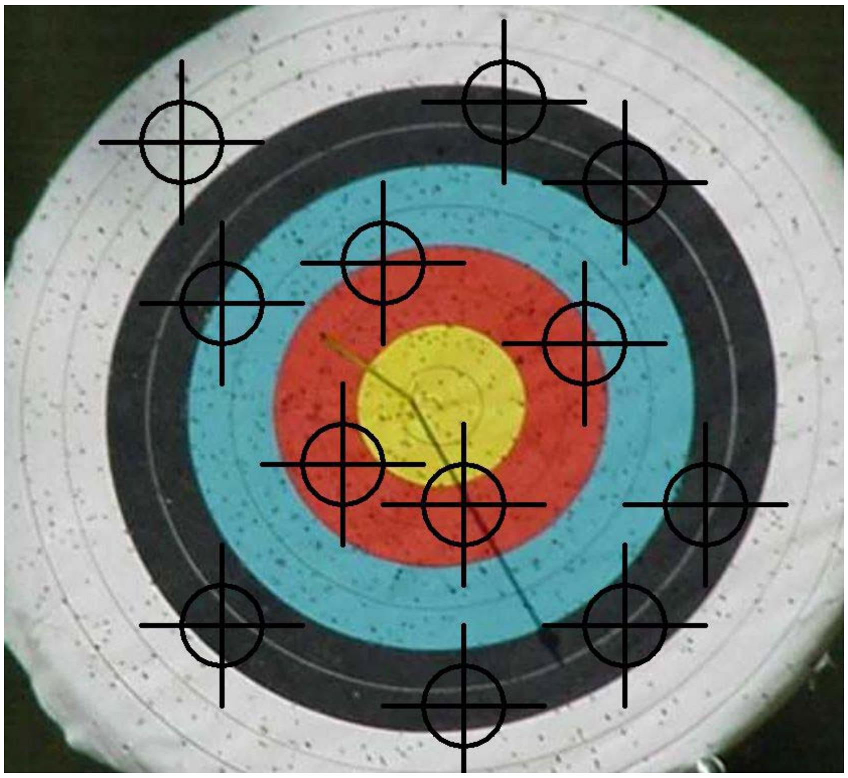


Figure 2.12: Piecewise constant actions (sightings)

2.5.4 Existing Approaches

There are a number of possible approaches for extending the Q-learning framework to continuous state and action spaces. The following sections briefly describe several methods. The categorisations are an attempt to capture the basic architecture of the methods and do not necessarily match the name of the algorithms used by the researchers.

Adaptive Critic methods

Werbos's (1992) adaptive critic family of methods use several feedforward artificial neural networks to implement reinforcement learning. The adaptive critic family includes methods closely related to Q-learning. A learned dynamic model assists in assigning reward to components of the action vector. Therefore, the approach does not meet the *Model-Free* criterion. If the dynamic model is already known, or learning one is easier than learning the controller itself, then model-based adaptive critic methods may be an efficient approach.

Q-AHC

Rummery's (1995) Q-AHC method that combines Q-learning with *Actor-Critic* learning. In Actor-Critic learning the *critic* component learns the expected value of *each state*, where the value is the expected reward over time from the environment under the behavioural policy (Barto et al., 1983). The critic component generates a local reward signal. The goal of the *actor* component is to maximise the local reward signal by choosing actions close to its current policy then changing its policy depending upon feedback from the critic. In turn, the critic adjusts the value of states in response to rewards received following the actor's policy. Actor-Critic systems can be easily adapted for continuous action learning; however, they are not capable of off-policy learning.

In Rummery's Q-AHC architecture Q-learning was used to choose between a set of continuous action Actor-Critic learners. Its performance overall was unsatisfactory since it generally either set the actions to constant settings, making it equivalent to Lin's (1992) system for generalising between states, or it only used one of the Actor-Critic modules, making it equivalent to a standard Actor-Critic system. These problems may stem from not fulfilling *Q Evaluation*, *Action Generalisation*, and *State Generalisation* criteria. The Q-AHC architecture is not capable of off-policy learning. A strong point of the method is that it is one of the few systems that can represent non-piecewise constant policies, thereby fulfilling the *Continuity* criterion.

CMAC-based $\mathcal{Q}$-Learning

Saito and Fukuda (1994) proposed a continuous state and action $\mathcal{Q}$-learning architecture using Albus's (1975) CMAC (Cerebellar Model Articulation Controller). The CMAC is a function approximation system that features spatial locality. It is a compromise between a look up table and a weight-based approximator. It can generalise between similar states, but the output is finally discretised, making it impossible to completely fulfil the *Continuity* criterion. In CMAC-based $\mathcal{Q}$-learning the inputs to the CMAC are the state and action and the output is the expected value. To find $\mathcal{Q}_{\max}$ this implementation requires a search across all possible actions, calculating the action-value for each, to find the highest value. This may not fulfil the *Action Selection* and *State Evaluation* criteria.

Another concern is that approximation resources are allocated uniformly across the state and action spaces. The result is that parts of the state-action space that are unimportant, or could be represented simply, are represented with the same amount of approximation resources as critically important parts of the state space. This makes scaling to larger problems difficult. Santamaria et al. (1998) mitigate uniform resource allocation by pre-distorting the state information using a priori knowledge so that more important parts of the state space receive more approximation resources. For example, the position on the target in the archery problem could be pre-distorted by a tansigmoidal function so that the approximation resources emphasise the centre of the target, where accuracy is most important.

Both Santamaria et al. (1998) and Saito and Fukuda (1994) evaluated the CMAC-based method on optimal control tasks in simulation with one-dimensional actions. The tasks included reward penalties for energy use and coarse motions.

Memory-Based $\mathcal{Q}$-Learning

Santamaria et al. (1998) also investigated memory-based schemes. Memory-based function approximators store example data points to create a mapping. This approach may not be fast enough to fulfil the *Action Selection* and *State Evaluation* criteria since finding an action-value required a search through the stored data points. The data points are added to the mapping incrementally, thereby allocating approximation resources non-uniformly.

Santamaria et al. stated that the memory-based approximator required less memory than the CMAC-based method but required more computational resources. The policy represented by the memory-based system was piecewise constant, thereby not fulfilling the *Continuity* criterion.

Wire Fitting

Baird and Klopf (1993) proposed an approach called *Wire Fitting* built around a locally weighted interpolation system. Each *wire* is a combination of an action vector and its expected value, both changing according to the state to generate a curve. Baird and Klopf proposed the use of a function approximator to generate the wires that encode the action-values. The main feature of this method is that the action with the highest value can be found quickly, without extensive searching, irrespective of function approximation method. Additionally, discontinuous policies and value functions can be represented. This method is the foundation for the continuous state and action method proposed in Chapter 3.

Baird and Klopf (1993) also proposed a more specific architecture based around Wire Fitting and a memory-based approximator. Many prototype points form a memory-based mapping between the state and the best known action and its value. To choose an action to carry out, the current state is compared to the states in the memory-based mapping and the closest match is found. The corresponding action is the best known action in that state. This means that the actions are always piecewise constant; thereby not fulfilling the *Continuity* criterion. When the resulting reward and next-state have been measured, a data point $(x, u, \mathcal{Q}(x, u))$ is generated and added to a buffer of points. The oldest data point in the buffer is discarded. The prototype points describing the best known actions are updated based on the buffer of data points. This system fulfils all of the continuous state and action criteria apart from the *Continuity* criterion.

Linearly Weighted Combination approach

The continuously valued $\mathcal{Q}$-learning approach proposed by Takahashi et al. (1999) used coarse discretisation of states and actions, then uses linear interpolation to represent intermediate states and actions. Like CMAC-based $\mathcal{Q}$-learning, Takahashi et al.'s method allocated approximation resources uniformly across the state and action space. It could be combined with the state pre-distortion approach (Santamaria et al., 1998) to allocate resources more efficiently if a priori information is available.

The method was evaluated with a real robot visual servoing task. The two-dimensional state representation was composed purely of features from the vision system. This would seem to create a perceptual aliasing problem because the robot's current velocity should have an influence on its behaviour. No reward penalty for coarse motion was used. The weighted combination function approximator could have a smoothing effect on the actions. As well as reducing the coarseness of motions the smoothing could help to implicitly represent the current velocity.

Although the emphasis in this thesis is on continuity and smoothness, in some situations it may be necessary for actions to change discontinuously in response to a smooth change in state. Takeda et al. (2000) showed that the inability to represent discontinuities can be a problem with the linearly weighted combination approach. Other continuous state and action algorithms may also have the same deficiency. Takeda et al. demonstrated the problem with a robot equipped with an omnidirectional vision sensor. If the robot was seeking a ball directly behind itself, it could turn either left or right to face the ball. However, using the weighted combination approach, the robot could perform the average of turning left and right, hence remaining stationary.

Takeda et al. added the ability to approximate discontinuities by adaptively adding new discretisation boundaries. The enhancement automatically allocated approximation resources where they are most needed. Candidates for new boundaries were selected through a heuristic: if the action of turning left with high torque had the highest expected value in a state, but turning right with high torque was most valuable in a neighboring state, then a new discretisation boundary was introduced for the state where the expected values of the actions were the same. With each new boundary added, the region of inaccurate approximation shrunk.

The procedure improved performance in Takeda et al.'s experiments. However, the method is rather task specific. Furthermore, no procedure for removing boundaries was provided, limiting the adaptability of the method if circumstances change.

Hedger

Smart and Kaelbling's (2000) *Hedger* system used a fixed number of points to approximate the action-values. Each approximation point stored a state, action, and expected value ordered set. The stored information in each point was updated as learning progressed, moving approximation resources to where they were needed. The value of a particular action in a state was calculated by locally weighted regression (LWR) based on the stored experiences. Finding the highest valued action required a time consuming iterative process. Hence, the *Action Selection* criterion was not fulfilled.

The Hedger system was demonstrated with a robot steering task in a corridor. The state information was the robot's steering angle and position relative to the corridor; and the distance to the end of the corridor. The robot's velocity was not included. The learning system controlled the steering but not the translational velocity. Reward was only provided at the end of the corridor. Smart and Kaelbling also successfully demonstrated the algorithm's off-policy learning capabilities. The Hedger algorithm's stability was enhanced by maintaining knowledge about which regions of the state-action space were believed to be approximated properly.

Smart and Kaelbling noted that comparing the combined state and action vectors using Euclidean distance was unsatisfying, and suggested that a different distance metric would be more appropriate. If a particular action in a state is compared to one of the stored experiences, and it matches the stored experience's action closely but the states are not similar, then by Euclidean distance the point is fairly similar. Therefore, the value of the particular action could be assessed wrongly.

Neural Field $\mathcal{Q}$-Learning

Gross et al. (1998) implemented a $\mathcal{Q}$-learning system based on dynamic neural fields. A neural vector quantiser (Neural Gas) clustered similar states. The output of the neural vector quantiser was the input to a neural field with lateral connections that encodes the values of actions. Finding the action with the highest expected value required iterative evaluation of the neural field dynamics. This may limit the speed with which actions can be selected (the *Action Selection* criterion) and values of states found (the *State Evaluation* criterion). The system fulfils the *State Generalisation* and *Action Generalisation* criteria.

The learning system was evaluated using a Khepera mobile robot. The robot's task was to dock using vision. It was a pure-delayed reward task as reward was only provided when docking was complete. After learning has been completed, task execution took around 12 steps, depending on the starting position.

The state representation did not include the robot's current velocity and the reward function did not include any penalties for coarse motions. The smoothing effect of the neural field dynamics might implicitly represent the current velocity and smooth movements. Additionally, the small size of the robot results in low momentum.

$\mathcal{Q}$-Self Organising Map

The $\mathcal{Q}$-KOHON system applied Kohonen's (1989) self organising map to $\mathcal{Q}$-learning (Sehad and Touzet, 1994; Touzet, 1997). The state, action, and expected value were the elements of the self organising map feature vector. The self organising map generalises between similar states and similar actions. Touzet suggests that it is possible to interpret the feature vectors in a self organising map-based implementation; whereas feedforward neural network-based systems are less transparent and are therefore harder to analyse.

The $\mathcal{Q}$-KOHON system was evaluated on a Khepera mobile robot. The robot learnt to avoid obstacles in a constructed environment using infra-red sensors (Touzet, 1997). There is no mention of penalties for coarse motion or the inclusion of current velocity in the state vector.

Actions were chosen by searching for the self organising map feature vector that most closely matched the state and the maximum representable value (one). The matching process has a potential flaw. If a state has a low expected value, then the matching process could select a feature vector that has a high value, rather than a feature vector that has a similar state vector. This could result in an action that is not appropriate for the current state.

The actions generated by the $\mathcal{Q}$-KOHON system are piecewise constant and, therefore, do not fulfil the *Continuity* criterion. For fairly smooth control the number of feature vectors must be high. Additionally, the number of feature vectors required must grow with the dimensionality of the state and action spaces. Choosing an action to perform requires a comparison to every feature vector. If the number of feature vectors is large the system would not fulfill criterion *Action Selection*.

$\mathcal{Q}$-Radial Basis

Santos (1999) described a system based on radial basis functions (Moody and Darken, 1989). It is similar to the $\mathcal{Q}$-KOHON system in that each hidden layer radial basis neuron holds a centre vector, much like the Kohonen self organising map feature vector. The number of possible actions is equal to the number of hidden layer radial basis neurons. Therefore, actions are piecewise constant and do not fulfil the *Continuity* criterion. An interesting property of the system is that it was initialised with zero hidden neurons; when new situations were encountered the number of hidden neurons increased.

The learning system was evaluated on a two degree of freedom robot arm that learnt to avoid virtual obstacles. The system was also evaluated with a Khepera mobile robot wall following task, and the same obstacle avoidance problem described for the $\mathcal{Q}$-KOHON system. The mobile robot was equipped with reflexes to protect it during learning. The wall avoidance behaviour was successfully learnt in 6 out of 10 trials with a learning time of under 45 minutes. For a larger robot in a natural environment the fairly long learning time would be taxing, unless the protective reflexes were exceptionally reliable.

The $\mathcal{Q}$-KOHON method does not meet the *$\mathcal{Q}$ Evaluation* criterion since only the actions described by the hidden radial basis neurons have an associated value. Accordingly, the system does not fully support off-policy learning.

Actor, $\mathcal{Q}$ methods

Several research groups have proposed methods that maintain an *actor* neural network, as in Actor-Critic learning (Barto et al., 1983), in addition to a neural network representing the action-values used in $\mathcal{Q}$-learning. The actor network directly provides the policy. Consequently, actions can be chosen quickly. The value of a particular state is calculated indirectly, by feeding the action produced by the actor network into the action-value network. The process assumes that the actor network is producing the action with the highest estimated value.

Munro (1987) proposed this system in the immediate reward context. Jordan and Jacobs (1990) described a similar system that can cope with delayed rewards. The 'forward model' they refer to is similar to the action-values of $\mathcal{Q}$-learning, rather than a forward dynamic model.

Maire (2000) described an actor, $\mathcal{Q}$ system using the framework of $\mathcal{Q}$-learning. Updates to the actor network were performed by feeding the current action through the action-value network, raising the expected value produced, and then backpropagating the change back through the action-value network to find the change in the action. The change in action was used to train the actor network. The purpose of the update process is for the actor network to seek higher value policies by performing gradient ascent on the action-value network.

The system fulfils all of the criteria presented in Figure 2.10 as long as the actor and action-value networks are synchronised. Synchronisation between the networks is required in order to fulfil criteria *Action Selection* and *State Evaluation*. The actor network is effectively an approximate caching mechanism. The action-value network alone represents sufficient information for $\mathcal{Q}$-learning. However, searching for the action with the highest value would be intolerably slow. Accordingly, the purpose of the actor network is to approximately cache the action with the highest expected value so that the time consuming search is not required. This method relies on the actor network successfully finding the appropriate action by gradient ascent on the action-value network.

Fuzzy $\mathcal{Q}$-Learning

Fuzzy controllers map continuous state into membership of discrete classes (*fuzzification*), then pass the membership through logical rules producing actions, and finally combine the actions based on the strength of their membership (*defuzzification*) (Zadeh, 1965). Fuzzy logic appears to be an ideal tool for creating a continuous state and action $\mathcal{Q}$-learning system: fuzzy logic generalises discrete rules for continuous data and $\mathcal{Q}$-learning can select discrete rules. A major difficulty is that fuzzy controllers execute many discrete rules simultaneously, complicating credit assignment.

Glorennec's (1994) system regarded every possible complete fuzzy controller as an action. Consequently, there was an enormous space of actions to be searched and generalisation between actions was limited. This does not satisfy criteria *Action Selection*, *State Evaluation*, and *Action Generalisation*. Bonarini (1996) treated individual rules as actions and searched through the space of rules using evolutionary methods. Berenji's (1994) system only fired one rule at a time. Consequently, $\mathcal{Q}$-learning can be applied more easily. The drawback is a reduced range of policies.

The fuzzy $\mathcal{Q}$-learning systems proposed to date do not satisfy the *$\mathcal{Q}$ Evaluation* criterion; thereby limiting off-policy learning. The number, shape, and bias of the fuzzification rules is also fixed, limiting the range of possible policies (criterion *Flexible Policy*). However, Fuzzy $\mathcal{Q}$-learning systems are capable of producing smoothly varying actions (criterion *Continuity*).

2.5.5 Evaluation of Existing Continuous State and Action Systems

There is a conflict between the desire to accurately represent the action-values and the desire to find the highest valued action as quickly as possible.

If a flexible, parameter-based approximator, such as an artificial neural network, were used, finding the maximum would require some iterative process (Gross et al., 1998; Smart and Kaelbling, 2000), or storing the policy separately (Jordan and Jacobs, 1990; Maire, 2000). Iterative processes can be time-consuming. Storing the policy separately could result in a policy that does not use optimal actions even if an optimal action is properly represented by the action-values.

Other systems used simple approximators in order to make a global search for the maximum of the action-values practical. However, the use of a simple approximator can introduce discretisation or otherwise make it difficult to adequately represent the action-values.

Baird and Klopf's (1993) Wire Fitting idea is unique in that it can be combined with any function approximator and provides a method for finding the highest valued action quickly, without requiring a separate policy network.

The main motivation for much of the work surveyed here is to produce smoothly varying actions. However, with the notable exception of the work of Takahashi et al. (1999) and Takeda et al. (2000) there are few published graphs, even from simulations, showing the action output. The position traces that are usually reported are insufficient to assess whether actions produced smooth robot motions.

It also seems strange that in the work surveyed only Saito and Fukuda (1994) and Santamaria et al. (1998) mention negative rewards for penalising coarse motions, or inclusion of velocity in the state vector. A control system formulation of a continuous state and action problem would almost always require some limit or cost to prevent coarse motions. Perhaps some of the systems surveyed explicitly generate smooth movements, and thereby reduce the need for penalties and state representation. Also, the use of small robots and large time intervals between actions reduces the effects of momentum and alleviates the perceptual aliasing problem. However, long intervals between actions are not suitable for robot interaction with dynamic environments. The time interval between actions was not documented in many of the works surveyed.

Finally, many of the learning systems described in this survey have not been tested on robots, making it difficult to assess whether there are any practical problems with the reported algorithms.

2.6 Summary

Reinforcement learning problems involve finding a policy that will maximise rewards over time. $\mathcal{Q}$-Learning is a model-free approach to solving reinforcement learning problems that is capable of learning off-policy. Off-policy learning techniques have the potential to contribute toward the parsimonious use of measured data—the most expensive resource in robotics.

Many robotics applications require smooth control: the ability to respond to a smooth change in state with a smooth change in action. Existing continuous state and action $\mathcal{Q}$-learning algorithms either lack smoothness, are incapable of off-policy learning, are slow to choose actions, or have not been tested on real robots.

This chapter identified the need for a continuous state and action $\mathcal{Q}$-learning algorithm suitable for robotics. Such an algorithm is the foundation for the research reported in this thesis.

Chapter 3

A Continuous State and Action Q-Learning Algorithm

A monk asked, "What about when there is neither a square nor a circle?"
The master said, "Neither square nor circle."
A monk asked, "When things are like that, what about it?"
The master said, "Either square or circle."

Joshu, c. 890, no. 275

This chapter presents an algorithm for continuous state, continuous action Q-learning. The algorithm is the fulfilment of an idea from Baird and Klopf (1993): it couples a single feedforward artificial neural network with an interpolator—*the wire fitter*. The method fulfils all the criteria for continuous state and action Q-learning set down in Chapter 2. It can also represents discontinuities in the policy and value function. The algorithm is progressively introduced through building on previous approaches, then illustrated with a simulation example.

3.1 Background

Feedforward artificial neural networks have been successfully used to generalise between similar states in Q-learning systems where actions are *discrete*. In the QCON architecture the state is the input to a set of neural networks, one network for each discrete action (Lin, 1992; Rummery, 1995). The output from each neural network is the expected value of performing its corresponding action. Alternatively, in the single network version of the architecture, multiple outputs of the single neural network represent the expected values of each discrete action. Such a discrete action system can generalise between similar states but not similar actions. The single network QCON architecture is defined in Figure 3.1.

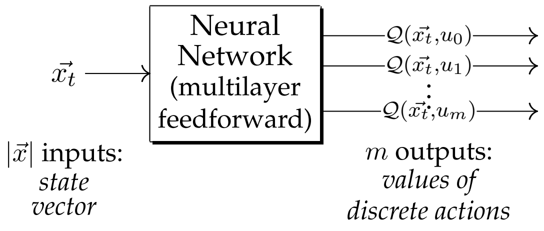

$\vec{x}_t$
Neural Network (multilayer feedforward)
$\mathcal{Q}(\vec{x}_t,u_0)$
$\mathcal{Q}(\vec{x}_t,u_1)$
$\mathcal{Q}(\vec{x}_t,u_m)$
$|\vec{x}|$ inputs: state vector
m outputs: values of discrete actions

Figure 3.1: Single network QCON architecture

An architecture that can generate continuously variable actions is shown in Figure 3.2. In this architecture the action vector is represented by a set of outputs from the neural network; that action's expected value is an additional output. The architecture represents the optimal action for any state as well as the value of that state. However, it does not support $\mathcal{Q}$-learning since it only represents the value of the optimal action, rather than all actions. Touzet (1997) calls it the 'ideal' architecture and points out the impracticality of a direct implementation.

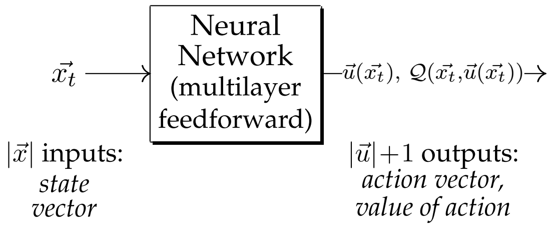

$\vec{x}_t$
Neural Network (multilayer feedforward)
$\vec{u}(\vec{x}_t),\ \mathcal{Q}(\vec{x}_t,\vec{u}(\vec{x}_t))$
$|\vec{x}|$ inputs: state vector
$|\vec{u}|+1$ outputs: action vector, value of action

Figure 3.2: 'Ideal' architecture. $\vec{u}\,(\vec{x})$ represents that the action vector is a function of the state vector. $\mathcal{Q}\,(\vec{x},\vec{u}\,(\vec{x}))$ is the value of action vector $\vec{u}\,(\vec{x})$ in state $\vec{x}$.

The 'ideal' architecture outputs an action vector and its value. If the output architecture were replicated several times the value of other action vectors could be represented (see Figure 3.3). It is a single network version of Rummery's (1995) $\mathcal{Q}$-AHC architecture described in Chapter 2. This architecture cannot generalise between actions.

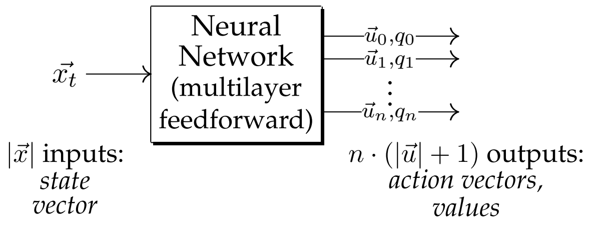

$\vec{x}_t$
Neural Network (multilayer feedforward)
$\vec{u}_0,q_0$
$\vec{u}_1,q_1$
$\vec{u}_n,q_n$
$|\vec{x}|$ inputs: state vector
$n\cdot(|\vec{u}|+1)$ outputs: action vectors, values

Figure 3.3: Single network $\mathcal{Q}$-AHC architecture. To condense the notation $\vec{u_i}$ represents $\vec{u_i}(\vec{x})$; and q_i or $q_i\,(\vec{x})$ represents $\mathcal{Q}\,(\vec{x},\vec{u_i}(\vec{x})\,,\vec{x})$.

3.2 The Wire Fitted Neural Network

In this research we propose the *wire fitted neural network* (WFNN) architecture, built upon an idea from Baird and Klopf (1993). This architecture is the foundation for our reported work.

The WFNN architecture adds an interpolator to the single network $\mathcal{Q}$-AHC architecture to represent the value of action vectors that do not exactly match outputs from the neural network. Following the terminology of Baird and Klopf, the combination of an action vector, $\vec{u_i}$, and its expected value, q_i, is called a *wire*. The term *wire* describes the curving variation of $\vec{u_i}$ and q_i in response to $\vec{x}$.

Baird and Klopf suggested the combination of approximator and interpolator, and described a suitable interpolation scheme. The *wire fitting interpolation function* is a moving least squares interpolator, closely related to Shepard's function (see Lancaster and Šalkauskas, 1986). Baird and Klopf used the wire fitting function in a memory-based reinforcement learning scheme introduced in Chapter 2. In the WFNN architecture the parameters describing wire positions are the output of a neural network. Figure 3.4 shows the architecture.

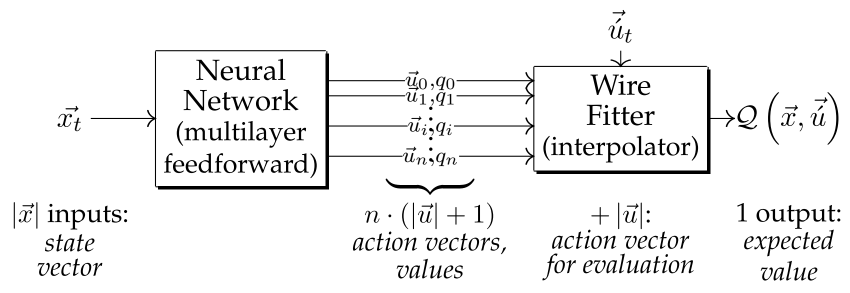

$\vec{u}_t$
$\vec{x}_t$
Neural Network (multilayer feedforward)
$\vec{u}_0, q_0$
$\vec{u}_1, q_1$
$\vec{u}_i, q_i$
$\vec{u}_n, q_n$
Wire Fitter (interpolator)
$\mathcal{Q}\left(\vec{x}, \vec{u}\right)$
$|\vec{x}|$ inputs: *state vector*
$n \cdot (|\vec{u}| + 1)$ *action vectors, values*
$+\,|\vec{u}|$: *action vector for evaluation*
1 output: *expected value*

Figure 3.4: WFNN architecture

The wire fitting function is (Baird and Klopf, 1993),

$$\mathcal{Q}\left(\vec{x}, \vec{u}\right) = \lim_{\epsilon \to 0^+} \frac{\sum_{i=0}^{n} \frac{q_i(\vec{x})}{\|\vec{u}-\vec{u_i}(\vec{x})\|^2 + c(\max_i q_i(\vec{x}) - q_i(\vec{x})) + \epsilon}}{\sum_{i=0}^{n} \frac{1}{\|\vec{u}-\vec{u_i}(\vec{x})\|^2 + c(\max_i q_i(\vec{x}) - q_i(\vec{x})) + \epsilon}} \tag{3.1}$$

where i is the wire number; n is the total number of wires; $\vec{x}$ is the state vector; $\vec{u_i}\,(\vec{x})$ is the ith action vector; $q_i\,(\vec{x})$ is the value of the ith action vector; $\vec{u}$ is the action vector to be evaluated; c is a small smoothing factor; and ϵ avoids division by zero. The dimensionality of the action vectors $\vec{u}$ and $\vec{u}_i$ is the number of continuous variables in the action.

The action that is expected to be optimal is $\arg\max_{\vec{u}} \mathcal{Q}\,(\vec{x}, \vec{u})$. This action must be quickly calculated to fulfil the *Action Selection* criterion. When choosing an action it is sufficient to propagate the state through the neural network, then compare the q outputs to find the action expected to be optimal. A property of the wire fitting interpolator is that the highest interpolated value always coincides with the highest valued interpolation point, so the action with the highest value is always one of the input actions: $\arg\max_{\vec{u}} \mathcal{Q}(\vec{x}, \vec{u}) = \vec{u}_{\arg\max_i q_i}$.

The wire fitting function has several other properties that make it a useful interpolator for implementing $\mathcal{Q}$-learning. Wire fitting works with many dimensional scattered data while remaining computationally tractable; no inversion of matrices or global search is required. Interpolation is local; only nearby points influence the value of $\mathcal{Q}$. Regions far from all wires have a value which is the average of $\vec{q}$; thus wild extrapolations do not occur (see Figure 3.5). Also, unlike most polynomial interpolation schemes, the method does not suffer from oscillations.

The WFNN architecture represents policies flexibly. The action represented by each wire changes smoothly in response to changes in the state, fulfilling the *Continuity* and *Flexible Policy* criteria. However, sudden changes in action are also possible by changing wires.

Updates to the action-values through Equation (2.4) require $\max_{\vec{u}} \mathcal{Q}(\vec{x}, \vec{u})$ (see the *State Evaluation* criterion). The maximum can be quickly found by propagating the *next-state* through the network then finding the maximum of the q outputs. Again, the wire fitter is not required. Use of this method causes a slight error in $\max_{\vec{u}} \mathcal{Q}(\vec{x}, \vec{u})$ since $\max_{\vec{u}} \mathcal{Q}(\vec{x}, \vec{u}) < \max_i q_i$. The error is visible in Figure 3.5 as a gap between the fitted line and the highest valued wire.

The combination of a neural network and wire fitting is required to store all of the action-values and fulfil the $\mathcal{Q}$ *Evaluation* criterion. The wire fitter interpolates between the examples of $\vec{u}$ and q produced by the neural network for $\vec{x}$ to approximate $\mathcal{Q}\left(\vec{x}, \vec{\acute{u}}\right)$ for an arbitrary $\vec{\acute{u}}$. As the wire fitter is purely an interpolator it contains no information or learned parameters. However, the wire fitter has a role in the neural network's learning process. The one-step $\mathcal{Q}$-learning update, Equation (2.4), produces a change in $\mathcal{Q}\left(\vec{x}, \vec{\acute{u}}\right)$; the change should be reflected as a change in the weights of the neural network.

Figure 3.5 shows an example of the update process. The action in this case is one dimensional, but the same process occurs with many dimensional actions. The example shows a graph of action versus value ($\mathcal{Q}$) *for a particular state*. The number of wires is fixed; the position of the wires changes to fit new data. Required changes are calculated

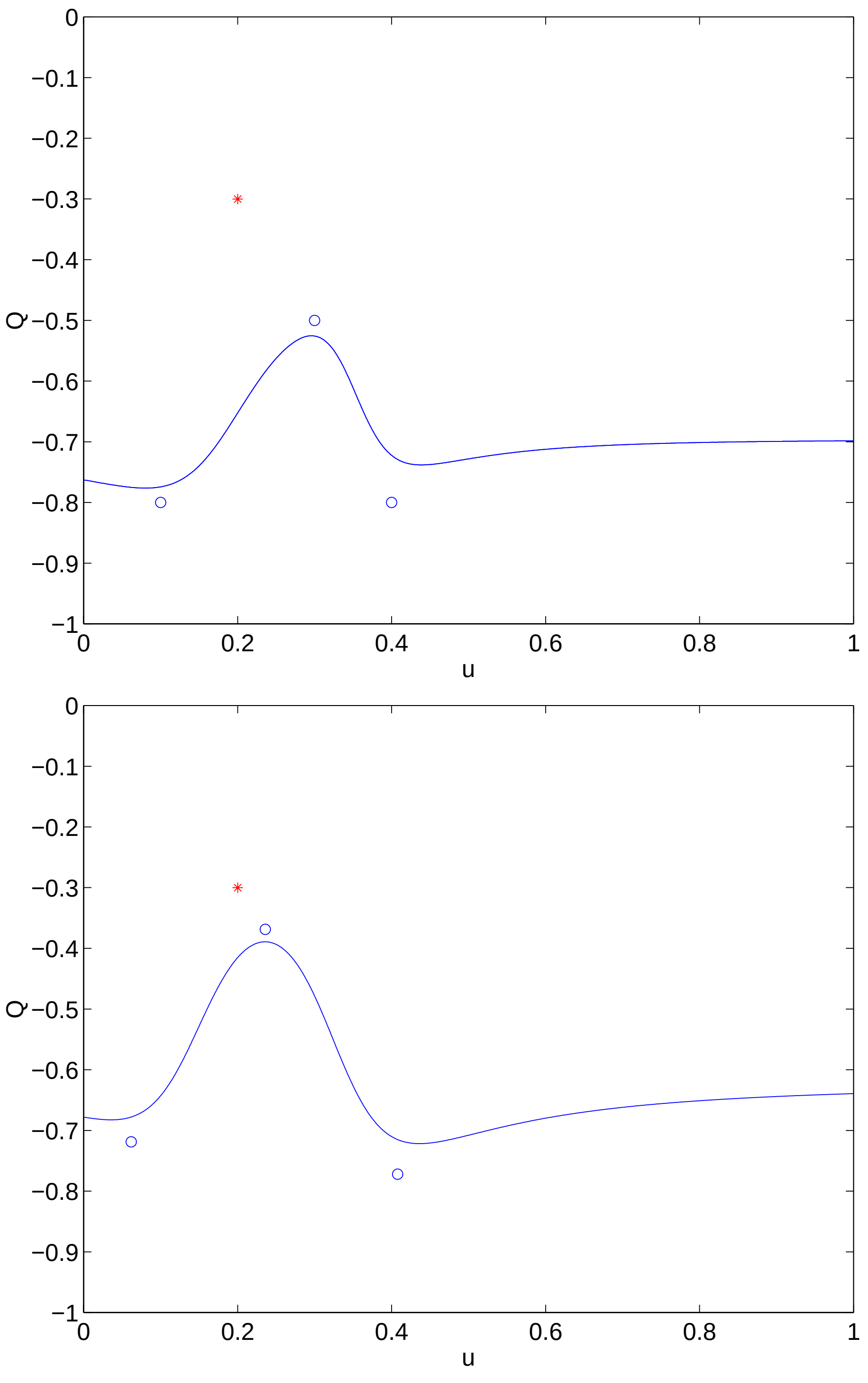


Figure 3.5: The wire fitting process. The action (u) is one dimensional in this case. Three wires (shown as $\circ$) are the output from the neural network for a particular state. The wire fitting function interpolates between the wires to calculate $\mathcal{Q}$ for every u. The new data ($*$) does not fit the curve well (*upper graph*); the wires are adjusted according to partial derivatives (*lower graph*). In other states the wires would be in different positions.

using the partial derivatives of the wire fitting function. Baird and Klopf (1993) specified the wire fitting function, but not the partial derivatives; their memory-based method did not require this information.

Partial derivatives in terms of each q and $\vec{u}$ of each wire can be calculated quickly. These partial derivatives allow the error in $\mathcal{Q}$ to be propagated to the neural network according to the chain rule:

$$\Delta q_i\left(\vec{x}\right) = \frac{\partial \mathcal{Q}\left(\vec{x}, \vec{u}\right)}{\partial q_i} \cdot \Delta \mathcal{Q}\left(\vec{x}, \vec{u}\right) \tag{3.2}$$

$$\Delta u_{i,j}\left(\vec{x}\right) = \frac{\partial \mathcal{Q}\left(\vec{x}, \vec{u}\right)}{\partial u_{i,j}} \cdot \Delta \mathcal{Q}\left(\vec{x}, \vec{u}\right) \tag{3.3}$$

where i is the wire number and j selects an action vector component ($\acute{u}_j$ is a component of the chosen action). No learning rate is necessary as it would as it would be absorbed by the neural network's learning rate. As a result of the change the $\mathcal{Q}$ output from the wire fitter moves closer to the new target $\mathcal{Q}$.

The partial derivative of $\mathcal{Q}\left(\vec{x}, \vec{u}\right)$ from Equation (3.1) in terms of $q_i\left(\vec{x}\right)$ is defined as,

$$\frac{\partial \mathcal{Q}\left(\vec{x}, \vec{u}\right)}{\partial q_i} = \lim_{\epsilon \to 0^+} \frac{norm\left(\vec{x}, \vec{u}\right) \cdot \left(distance\left(\vec{x}, \vec{u}\right) + q_i \cdot c\right) - wsum\left(\vec{x}, \vec{u}\right) \cdot c}{\left[norm\left(\vec{x}, \vec{u}\right) \cdot distance\left(\vec{x}, \vec{u}\right)\right]^2} \tag{3.4}$$

where

$$\begin{aligned}
distance\left(\vec{x}, \vec{u}\right) &= \|\vec{u} - \vec{u_i}\left(\vec{x}\right)\|^2 + c\left(\max_i q_i\left(\vec{x}\right) - q_i\left(\vec{x}\right)\right) + \epsilon \\
wsum\left(\vec{x}, \vec{u}\right) &= \sum_{i=0}^{n} \frac{q_i\left(\vec{x}\right)}{\|\vec{u} - \vec{u_i}\left(\vec{x}\right)\|^2 + c\left(\max_i q_i\left(\vec{x}\right) - q_i\left(\vec{x}\right)\right) + \epsilon} \\
norm\left(\vec{x}, \vec{u}\right) &= \sum_{i=0}^{n} \frac{1}{\|\vec{u} - \vec{u_i}\left(\vec{x}\right)\|^2 + c\left(\max_i q_i\left(\vec{x}\right) - q_i\left(\vec{x}\right)\right) + \epsilon}
\end{aligned}$$

Using these simplifications the wire fitting equation (3.1) becomes,

$$\begin{aligned}
\mathcal{Q}\left(\vec{x}, \vec{u}\right) &= \lim_{\epsilon \to 0^+} \frac{\sum_{i=0}^{n} \frac{q_i(\vec{x})}{distance(\vec{x}, \vec{u})}}{\sum_{i=0}^{n} \frac{1}{distance(\vec{x}, \vec{u})}} \\
&= \lim_{\epsilon \to 0^+} \frac{wsum\left(\vec{x}, \vec{u}\right)}{norm\left(\vec{x}, \vec{u}\right)}
\end{aligned}$$

Equation (3.4) is inexact when $i = \arg\max_i q_i$ (for the wire with the highest q).

The partial derivative of $\mathcal{Q}\left(\vec{x}, \vec{u}\right)$ in terms of $u_{i,j}(\vec{x})$ is defined by,

$$\frac{\partial \mathcal{Q}\left(\vec{x}, \vec{u}\right)}{\partial u_{i,j}} = \lim_{\epsilon \to 0^+} \frac{\left[wsum\left(\vec{x}, \vec{u}\right) - norm\left(\vec{x}, \vec{u}\right) \cdot q_i\right] \cdot 2 \cdot \left(u_{i,j} - \acute{u}_j\right)}{\left[norm\left(\vec{x}, \vec{u}\right) \cdot distance\left(\vec{x}, \vec{u}\right)\right]^2} \tag{3.5}$$

The summation terms in Equations (3.4) and (3.5) have already been determined in the calculation of $\mathcal{Q}\left(\vec{x}, \vec{u}\right)$ with Equation (3.1).

After the changes to each q and $\vec{u}$ have been calculated the neural network must be trained to output the new values. Training is performed through standard incremental backpropagation (Rumelhart et al., 1986). Every weight of the neural network is updated to bring the neural network's output closer to the target output.

We used tansigmoidal neurons throughout the neural network. Consequently, the output of the neural network is limited to between -1 and 1 by the hyperbolic tangent function. Therefore, only action-values within this range can be represented. If the reward signal requires action-values outside the range then divergence will occur. For reward signals scaled between -1 and 1 it is necessary to pre-scale the reward by multiplying by $1 - \gamma$ to keep the action-values within the representable range.

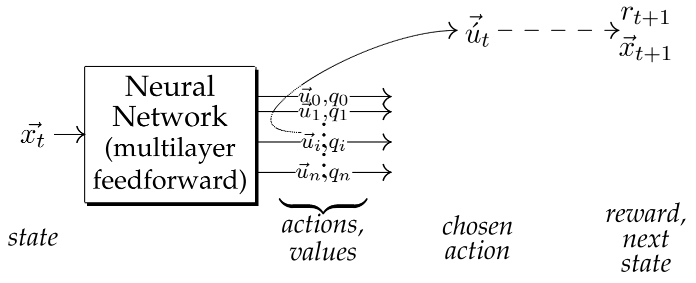
Nevertheless, if the unscaled reward had a magnitude higher than 1 then the required action-value will still be outside the allowed range after scaling. To prevent this from occurring the reward is limited by the hyperbolic tangent function before scaling. Thus,

$$r = (1 - \gamma) \tanh(\text{untransformed reward}) \tag{3.6}$$

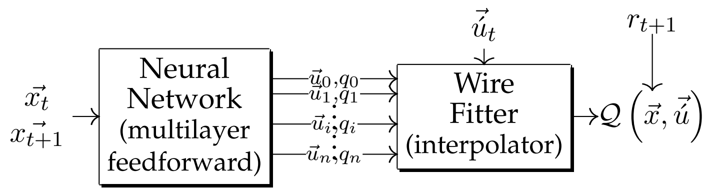
The transformation of the reward signal is implicitly applied in all of the learning tasks in this thesis.

3.3 Training Algorithm

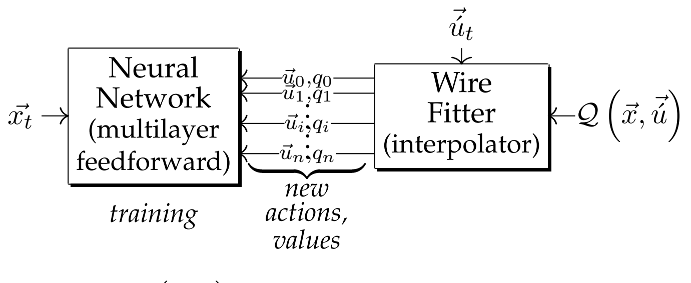
The WFNN training algorithm is shown in Figure 3.6, and in more detail in Figures 3.7, 3.8, and 3.9. If learning is judged to be adequate and no further learning is required then only Part 1 of the algorithm is necessary. Parts 1 and 2 do not require the wire fitter. Parts 2 and 3 can be executed as a separate process to Part 1. The only communications between the two processes are: the experiences generated by Part 1 are sent to Parts 2 and 3; and the neural network weights calculated by Parts 2 and 3 are sent to Part 1.



1. Feed the state into the neural network. From the output of the neural network find the action with the highest q. Execute the action ($\vec{u}$). Record an experience composed of the initial state, action, next-state, and the reward received as a result of the action. This step must be performed in real-time.

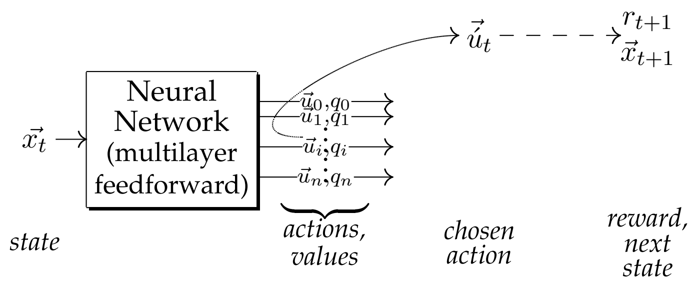


2. Calculate a new estimate of $\mathcal{Q}\left(\vec{x}, \vec{u}\right)$ from an experience composed of a state, action, next-state, and reward using the one step $\mathcal{Q}$-learning formula, Equation (2.4). This can be done when convenient or off-line.



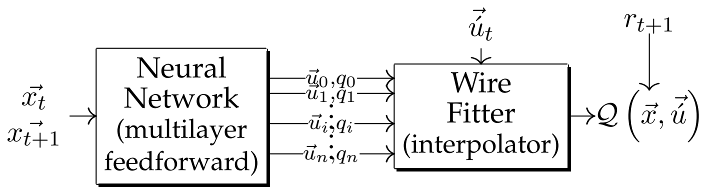
3. From the new value of $\mathcal{Q}\left(\vec{x}, \vec{u}\right)$ calculate new values for $\vec{u}$ and $\vec{q}$ through Equations (3.2) and (3.3), using the wire fitter partial derivatives, Equations (3.4) and (3.5). Train the neural network to output the new $\vec{u}$ and $\vec{q}$. This can also be done when convenient or off-line.

Figure 3.6: The WFNN training algorithm, Parts 1,2, and 3



1. To be executed in real-time:
   (a) set the time, $t = 0$;
   (b) measure the current state, $\vec{x}_t$;
   (c) pass $\vec{x}_t$ through the neural network to produce $\vec{u}$ and $\vec{q}$;
   (d) as
   $$\arg\max_{\vec{u}} \mathcal{Q}(\vec{x}, \vec{u}) = \vec{u}_{\arg\max_i q_i}$$
   select action:
   $$\vec{u}_t \leftarrow \vec{u}_{\arg\max_i q_i} \quad ;$$
   (e) optional step: to encourage exploration, add noise to $\vec{u}_t$;
   (f) execute action $\vec{u}_t$;
   (g) wait until next time tick;
   (h) measure the next-state, $\vec{x}_{t+1}$, and reward, r_t;
   (i) if $\vec{x}_{t+1}$ is not a terminating state:
   store the experience: $\vec{x}_t, \vec{u}_t, \vec{x}_{t+1}, r_t$;
   otherwise store: $\vec{x}_t, \vec{u}_t, r_t$;
   (j) update:
   $$\vec{x}_t \leftarrow \vec{x}_{t+1},$$
   $$t \leftarrow t + 1 \quad ;$$
   (k) repeat from Step 1c.

Figure 3.7: Part 1 of the WFNN training algorithm in greater detail

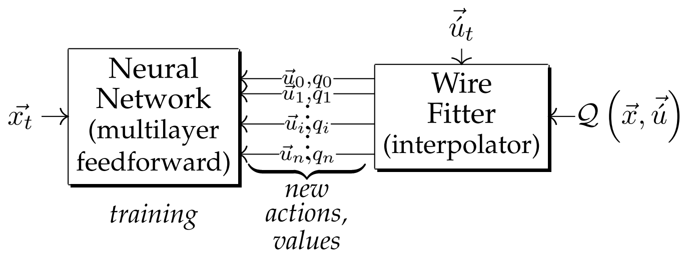


2. To be executed when convenient or off-line:
   (a) select an experience—either the last experience recorded, or a randomly chosen experience for experience replay—and set $t, \vec{x}_t, \vec{u}_t, \vec{x}_{t+1}, r_t$ appropriately;
   (b) if $\vec{x}_{t+1}$ exists:
      i. pass $\vec{x}_{t+1}$ through the neural network to produce $\vec{q}'$,
      ii. as $\max_{u_{t+1}} \mathcal{Q}\left(x_{t+1}, u_{t+1}\right) = \max_i q_i'$, set:

$$\mathcal{Q}_{new}\left(\vec{x}_t, \vec{u}_t\right) \leftarrow r_t + \gamma \max_i q_i' \quad \text{, as per Equation (2.4);}$$

   otherwise:
      set $\mathcal{Q}_{new}\left(\vec{x}_t, \vec{u}_t\right) \leftarrow r_t$

Figure 3.8: Part 2 of the WFNN training algorithm in greater detail



3. To be executed when convenient or off-line:
   (a) calculate the current estimate of $\mathcal{Q}\left(\vec{x}_t, \vec{u}_t\right)$:
      i. pass $\vec{x}_t$ through the neural network to produce $\vec{u}$ and $\vec{q}$, and
      ii. calculate $\mathcal{Q}\left(\vec{x}_t, \vec{u}_t\right)$ using Equation (3.1);
   (b) calculate the change in $\vec{u}$ and $\vec{q}$:
      i. set:

$$\Delta\mathcal{Q}\left(\vec{x}_t, \vec{u}_t\right) \leftarrow \mathcal{Q}_{new}\left(\vec{x}_t, \vec{u}_t\right) - \mathcal{Q}\left(\vec{x}_t, \vec{u}_t\right)$$

      ii. calculate $\frac{\partial \mathcal{Q}(\vec{x}_t, \vec{u}_t)}{\partial q_i}$ for all i using Equation (3.4),
      iii. calculate $\frac{\partial \mathcal{Q}(\vec{x}_t, \vec{u}_t)}{\partial u_{i,j}}$ for all i, j using Equation (3.5), then
      iv. set $\vec{u}_{new}$ and $\vec{q}_{new}$ according to Equations (3.2) and (3.3);
   (c) train the neural network to produce $\vec{u}_{new}$ and $\vec{q}_{new}$ in response to $\vec{x}_t$

Figure 3.9: Part 3 of the WFNN training algorithm in greater detail

3.4 Illustrating Example: Navigating a Boat to a Target Position

In this example application of the WFNN algorithm the task is to guide a simulated boat to a target position by firing thrusters located on either side. The thrusters produce continuously variable force ranging from full forward to full backward. As there are only two thrusters (left, right) but three degrees of freedom (x, y, rotation) the boat is non-holonomic in its planar world. The simulation includes momentum and friction effects in both angular and linear velocity. The controller must learn to slow the boat and hold position as it reaches the target. Reward is the negative of the distance to the target. This is a *delayed* reward problem, not a *pure-delayed* reward problem.

The dynamics of the boat are defined by the following equations:

$$\begin{aligned}\ddot{\theta} &= 0.1\,(A-B) - 0.15\dot{\theta} \\ \ddot{x} &= 0.1\,(A+B) - 0.15\dot{x} \\ \ddot{y} &= -0.15\dot{y}\end{aligned} \tag{3.7}$$

where θ is rotation and x and y are positions in body fixed coordinates. Position and rotation are integrated using the Euler method with a time step of 1. A and B are the thruster settings, between -1 and 1. The numerical constants in Equation (3.7) were chosen empirically. Momentum was set high and friction was set low to emphasise the need to plan ahead. Consequently, the boat must start slowing down long before it reaches the target. It is also easy to over-steer. Figure 3.10 describes the state and action representations and the reward equation for the boat navigation task.

State	x', y': relative position of target $\dot{x}$, $\dot{y}$: boat linear velocity $\dot{\theta}$: boat angular velocity
Action	A, B: left and right thrust
Reward	$-0.75\sqrt{x'^2 + y'^2}$: to encourage movement to the target

Figure 3.10: Reinforcement learning specification for boat navigation

Negative rewards are more successful than positive rewards since they encourage exploration. Scaling of the reward was set so that when a target is 2 units away reward is approximately -0.9. If the reward moves to -1 too quickly then the boat controller will receive no feedback if it wanders far from the goal. As discussed earlier, the reward is also limited by the hyperbolic tangent function, then rescaled, based on the discount factor.

In order to find worthwhile actions some form of exploration is necessary. In this example, exploration was achieved by performing actions drawn from the uniform random distribution 10% of the time.

3.4.1 Results

Figure 3.11 shows a simulation run with hand placed targets. At the point marked '0', near the center of the image, the learning system had no knowledge of the effects of its actuators, or the relationship between its sensor information and the task to be achieved. After some initial wandering the controller gradually learnt to guide the boat directly toward the target and come to a near stop.

Figure 3.12 shows the percentage of boats reaching the n^{th} target having reached the $(n-1)^{th}$ target and the absolute percentage of boats reaching the n^{th} target. In these experiments the controller learns to direct the boat to the first randomly placed target about 70% of the time. Less than half of the controllers reached all 10 targets. Figure 3.13 shows the number of steps taken for successful boats to reach the n^{th} target; the performance of the system improved with time. The simulation results shown are from 300 trials.

The characteristics of the learned controller are a function of the reward function and the simplicity of description of the controller. Use of the reward function in Figure 3.10 usually resulted in a controller that continuously spins the boat, using the following control approach:

- fix one thruster to full forward; and
- set the other thruster to full backward, but if the target is directly ahead set to full forward.

This controller, or one of its isomorphisms, is quickly learnt because it receives reward and is easily represented by the network.

To promote smooth control, and avoid the spinning controller, it is necessary to punish for energy use and sudden changes in commanded action through a modification to the reward function:

$$r = -0.5\sqrt{x'^2 + y'^2} - 0.5\left(A^2 + B^2\right) - 2\left((A-a)^2 + (B-b)^2\right) \tag{3.8}$$

where a, b are the previously commanded left and right thrust. These must be included in the state vector.

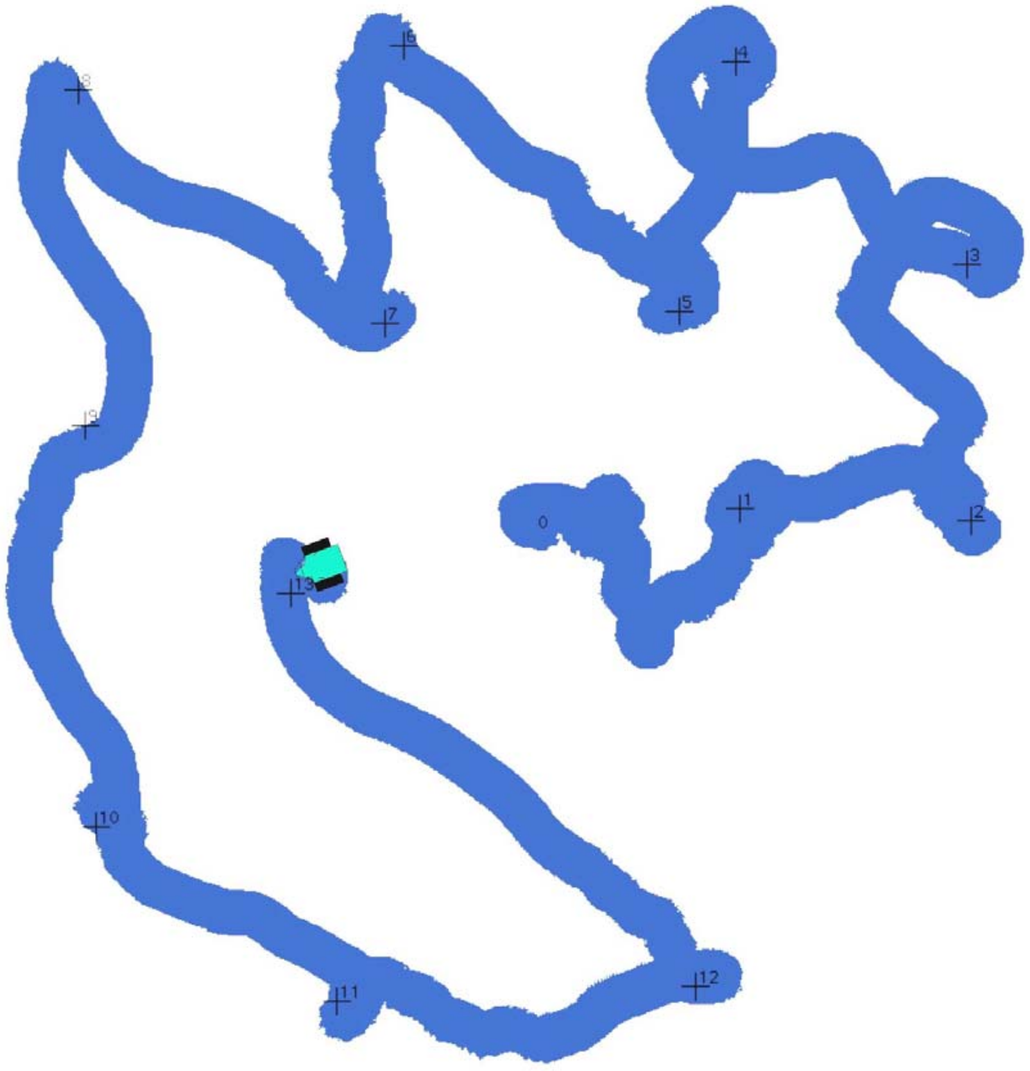


Figure 3.11: Appearance of the simulation for one run. The boat begins at the point marked 0. Targets are marked with a + symbol and target number. The boat gradually learnt to control its motion and move more directly to targets.

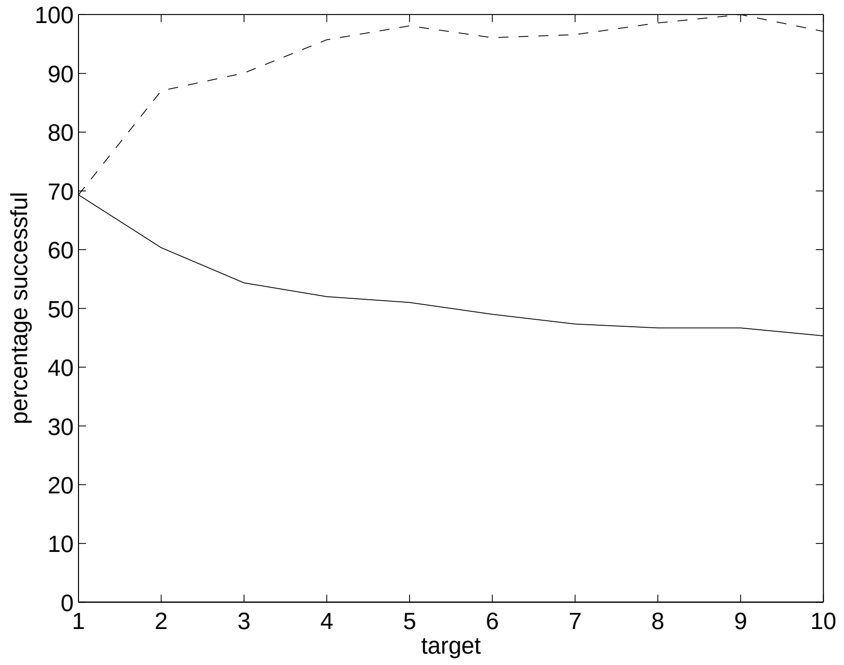


Figure 3.12: Performance of WFNN $\mathcal{Q}$-learning applied to the boat navigation task. Dashed line is percentage of simulated boats reaching target n after reaching target $n-1$; solid line is percentage of boats reaching target n.

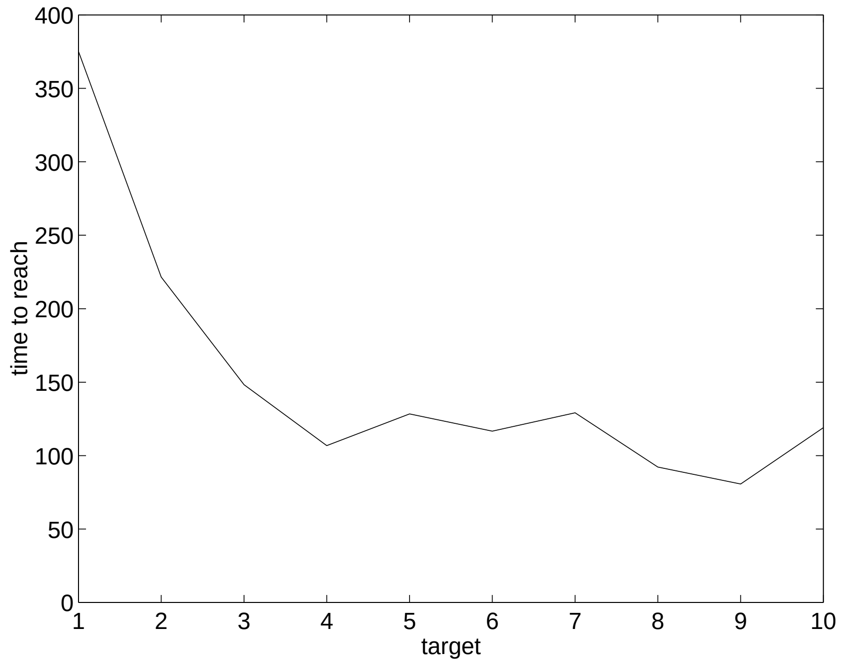


Figure 3.13: Number of action steps taken for the simulated boat to reach target n after reaching target $n - 1$.

The reward function including penalties in Equation (3.8) encouraged smoothness and confirmed that the system is capable of responding to continuous changes in state with continuous changes in action. An added benefit of punishing for consuming energy was an improved ability to maintain a fixed position. The experiment shows the importance of selecting an appropriate reward function.

Figure 3.14 shows the evolution of the action-values that describe the controller[1]. These graphs should be compared to the example graph in Figure 3.5. Figure 3.5 is for a reinforcement learning problem with a one dimensional action whereas Figure 3.14 is for a two-dimensional action vector. Each graph shows the portion of the action-values for a particular state. The upper graph of Figure 3.14 is at the start of learning; the policy is almost random at this stage. The wires are visible in the graph as peaks and basins in the action-values. The lower graph shows the portion of the action-values for a different state after the controller has converged.

3.4.2 Application of Off-Policy Learning

We applied two of the off-policy learning techniques described in Chapter 2 in solving the boat navigation task: *experience replay*, and *learning from phantom targets*.

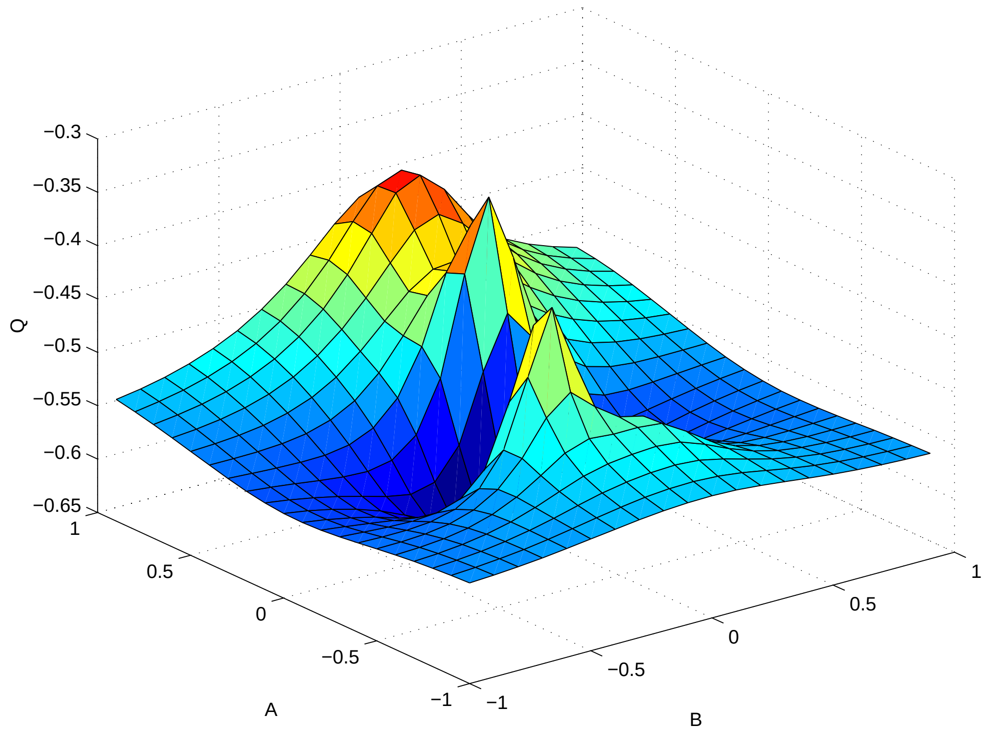
Experience replay increased learning speed by storing experiences in a database and replaying them repeatedly. The database of experiences was erased each time a new controller was learnt to avoid interference between trials.

The *learning from phantom targets* technique increased learning speed by allowing the learning system to learn from many phantom targets, rather than only the single, current target. Phantom targets were generated when experiences were replayed by modifying the position of the target by a random distance. In order to confirm that the system is capable of learning from the phantom targets the current (or real) target was never used by the learning process. Therefore, the algorithm must have learnt purely off-policy.

3.5 Summary

A continuous state and action Q-learning system has been proposed that fulfils the criteria in Chapter 2. State and action generalisation are performed through a combination of a feedforward neural network and an interpolator. Action selection can be performed without search, caching, or iteration. A disadvantage of WFNN is its ungainly

[1]The graphs in Figure 3.14 are for advantage learning, not Q-learning. Advantage learning will be described in Chapter 4. The graphs are included here in order to illustrate the operation of the wire fitter.



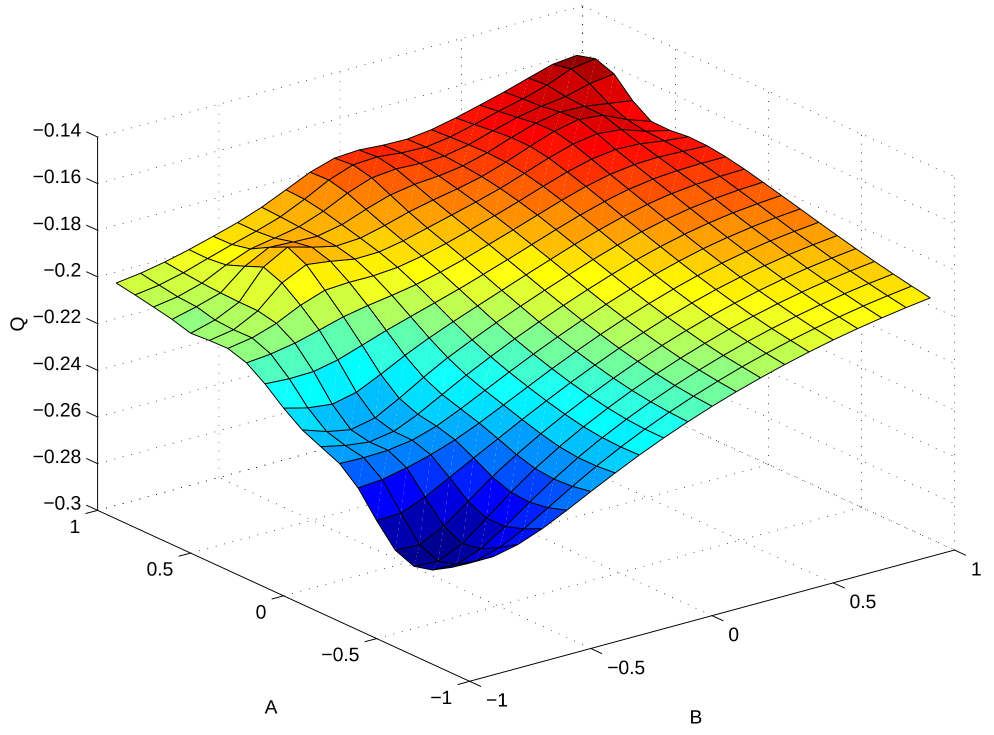


Figure 3.14: Graphs showing the portion of the action-values for a particular state. This is a surface fitted over several wires by the wire fitter. Axes A and B are the thruster command (u); the vertical axis is the expected value (Q). The *upper graph* is at the start of learning; in the *lower graph* learning has converged.

architecture. It is somewhat unattractive that two interpolators, the neural network and the wire fitter, are required, and that there are many configurable parameters. However, the approach is computationally inexpensive and allows representation of discontinuous as well as smoothly changing policies. The ability to learn off-policy was demonstrated through experience replay and learning from phantom targets.

Applying the learning system to a simulated task showed the importance of choosing an appropriate reward function. When the structure of the task and the reward function did not encourage smoothly changing actions the learned controller produced only coarse actions. Adding energy and coarseness penalties to the reward function guided the learning system towards smoother control.

The simulation results were promising; however, a significant proportion of controllers failed to converge. The following chapter discusses the stability and performance of the WFNN algorithm, and $\mathcal{Q}$-learning with continuous states and actions in general.

Chapter 4

Issues for Continuous $\mathcal{Q}$-Learning

A monk asked, "I don't have a special question. Please don't give a special reply."
The master said, "How extraordinary."

Joshu, c. 890, no. 376

The previous chapter described the WFNN continuous state and action $\mathcal{Q}$-learning system. A simulation experiment revealed that the learning controller does not always converge successfully. In this chapter we investigate the performance of the algorithm, including its sensitivity to changes in the design parameters. Statistical experiments show that significant improvements are possible through modifications to the basic algorithm. Some of the problems addressed and solutions proposed are relevant to function approximation-based $\mathcal{Q}$-learning systems in general. In particular, a partial solution to the rising $\mathcal{Q}$ problem is proposed.

4.1 Convergence

In the discrete states and actions case the value function can be represented exactly in a table. Convergence to the optimal controller is guaranteed, providing that there is a nonzero probability of executing every action in each state and the learning rate is decreased with a suitable schedule (Watkins and Dayan, 1992). In the continuous state case it is not possible to exactly store the value function, instead it is approximated. Approximation forfeits the guarantee of convergence. The simulation experiment in Chapter 3 demonstrated that the WFNN algorithm does not always converge. Furthermore, divergence can still occur even after a successful policy has been found. Numerous previous works give examples of situations where function approximator-based $\mathcal{Q}$-learning fails to converge, and a few suggest methods of managing the problem (Baird, 1995; Tsitsiklis and Roy, 1997; Thrun and Schwartz, 1993; Gordon, 1999; Boyan and Moore, 1995). The partial solutions proposed are only suitable for specialised approximators, or they make learning unacceptably slow.

Smart and Kaelbling (2000) improved stability by bounding the regions of reliable approximation with a convex hull. Not all Q-learning methods support this operation; in particular it would be difficult to apply the technique to the WFNN method.

Santamaria et al.'s (1998) results suggest several practical ways of improving the convergence of Q-learning systems with function approximation:

1. approximators with spatial locality to avoid the unlearning problem;
2. reducing the reliance on bootstrapping;
3. an intermediate value of λ; and
4. on-line learning so that the distribution of states for learning is similar to the distribution of states when executing the policy.

The first idea addresses the issue of *unlearning* that can occur in neural networks. When the input to a neural network covers only a small region of the input space the network can 'forget' the correct output for other regions of the state space. It is also known as *interference, fixation*, or *global network collapse* (White and Sofge, 1992). In control theory it is known as *bursting due to lack of persistent excitation* (Anderson, 1985). Unlearning can easily occur in a neural network-based reinforcement learning system as the input state is relatively static for long periods of time. Replaying experiences can provide a broader range of inputs, but repetitive training on the same data can still lead to over-training—the network fits the training data exactly but loses the ability to smoothly interpolate to new data (Lin, 1992).

Santamaria et al.'s (1998) approach to coping with the unlearning problem is to use an approximator with spatial locality. A feedforward neural network is a *global approximator*: changes to the neural network weights have some effect over the entire input space. In contrast, a local approximator's parameters affect only a small region of the input space. This alleviates the unlearning problem when the state only varies over a small range of values since the action-values for other areas of the state space should be preserved.

Local approximators generalise between similar inputs, but they are not capable of generalising at the next level—representing relationships between variables. A global approximator can represent higher relationships, for example, that the action-values in a particular problem are nearly linearly related to one of the state variables. High memory consumption can also be a consequence of local approximation since local approximators usually allocate their approximation resources uniformly. Santamaria et al. address the issue by pre-distorting the state and action variables based on a priori knowledge. It allocates more approximation resources to more important regions, but requires that knowledge be specified in advance. We favoured global approximation to make higher level generalisation possible and perhaps allow scaling to higher dimensional state spaces than would be possible with a local approximation method.

Santamaria et al. (1998) suggest improving convergence by reducing the reliance on bootstrapping. Chapter 2 described how the update equation for pure one-step Q-learning relies heavily on representing values accurately, whereas other algorithms mix represented and measured information and are more reliable when the representation is poor.

On-line learning is also suggested by Santamaria et al.'s results. In on-line learning, the value function is updated immediately as actions are executed and states are encountered. It leads to efficient use of approximation resources as the approximator should emphasise states and actions that are actually encountered. Some examples of divergence in reinforcement learning have been based on off-line model based updates that do not reflect the distribution of states encountered when executing the policy (Boyan and Moore, 1995).

This thesis focuses on one-step Q-learning without strict on-line learning because of the potential of passive learning techniques for robotics. Chapter 2 described how passive learning requires one-step Q-learning's ability to learn from off-policy actions.

4.2 The State-Action Deviation Problem

A first step in understanding the behaviour of WFNN in the simulation experiment is to visualise a portion of the action-values as shown in the graph in Figure 4.1. The graphs were not produced by pure Q-learning. After convergence to reasonable performance, pure Q-learning would produce a flat graph: the difference in value between actions in a particular state is almost imperceptible. If there is no difference in value between actions then there is no policy: the learning system does not express a preference for one action over another.

The similarity in values of actions occurs in control problems because carrying out

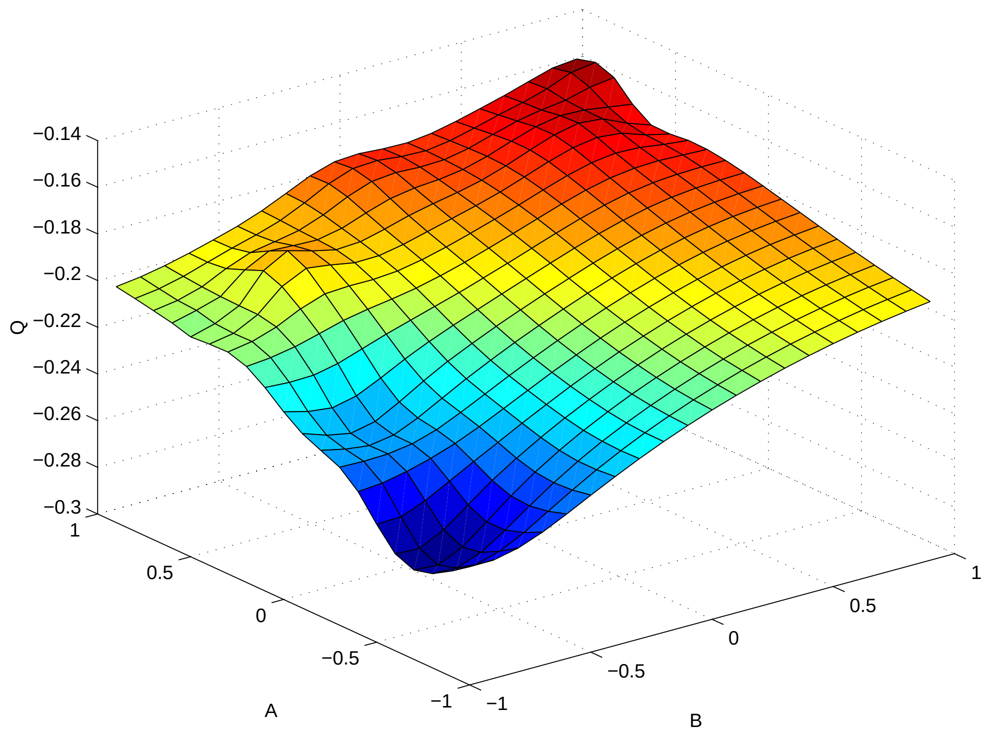


Figure 4.1: Graph showing the portion of the action-values for a particular state. The vertical axis is the expected value (Q). Under pure Q-learning the surface would be flat, representing no difference between the values of actions. (This is a reproduction of the lower graph in Figure 3.14.)

a single poor action rarely leads to an immediate disaster. In the next time step a good action can be chosen, and control can be maintained. In Q-learning, the update equation assumes that the best known action will be executed at the next time step, and the values of executing actions are carried backwards through time. Thus the values of all non-disastrous actions tend to be similar. The flattening effect increases as the time intervals between control actions are reduced.

If the action-values are only stored approximately it is likely that approximation resources will be used to represent the values of the states rather than actions in states. The relative values of actions will be poorly represented, resulting in an unsatisfactory policy. The problem was identified by Asada et al. (1994) and named the *state-action deviation problem*. Independently, Baird (1994) also identified the problem. The state-action deviation problem has been observed by other researchers who have used approximators to store the value function (Gross et al., 1998; Maire, 2000).

The state-action deviation problem mainly occurs in control-like problems. In more logical problems, such as playing chess, single actions can make a great deal of difference to the situation, and there are many disastrous actions. Asada et al. (1994) addressed the state-action deviation problem by treating the robot control problem as an event-based problem. Their system continued to execute an action until a change in discretised state was observed (Asada et al., 1994). Takahashi et al. (1999) used a similar approach with finely discretised states. However, such an event-based approach cannot produce actions that vary smoothly with the state.

Another approach to the state-action deviation problem would be to move away from the discrete time formulation of reinforcement learning and work in continuous time (Werbos, 1992; Baird, 1994; Doya, 1995; Bertsekas and Tsitsiklis, 1996). The continuous time formulation is not as well understood or explored as the discrete time formulation.

Harmon and Baird's (1995) *advantage learning* addresses the state-action deviation problem by emphasising the differences in value between the actions. In advantage learning the value of the optimal action is the same as for $\mathcal{Q}$-learning, but the lesser value of non-optimal actions is emphasised by a scaling factor ($k \propto \Delta t$). This makes more efficient use of the approximation resources available.

The advantage learning update is:

$$\begin{aligned}\mathcal{A}(x_t, u_t) \xleftarrow{\alpha} \tfrac{1}{k}\left[r(x_t, u_t, x_{t+1}) + \gamma \max_{u_{t+1}} \mathcal{A}(x_{t+1}, u_{t+1})\right]& \\ + \left(1 - \tfrac{1}{k}\right) \max_{u_t} \mathcal{A}(x_t, u_t)& \quad (4.1)\end{aligned}$$

This can be rearranged to emphasise the function of the scaling factor:

$$\begin{aligned}\mathcal{A}(x_t, u_t) \xleftarrow{\alpha}\ & r(x_t, u_t, x_{t+1}) + \gamma \max_{u_{t+1}} \mathcal{A}(x_{t+1}, u_{t+1}) \\ & - \left(1 - \tfrac{1}{k}\right) \cdot \left[r(x_t, u_t, x_{t+1}) + \gamma \max_{u_{t+1}} \mathcal{A}(x_{t+1}, u_{t+1}) - \max_{u_t} \mathcal{A}(x_t, u_t)\right] \quad (4.2)\end{aligned}$$

where $\mathcal{A}$ is analogous to $\mathcal{Q}$ in the update equation for $\mathcal{Q}$-learning.

The graph in Figure 4.1 was produced through advantage learning; the difference in value between actions is clear. Advantage learning improves the performance of the WFNN algorithm on the boat navigation task. Under advantage learning 100% of the controllers converge to acceptable performance, versus 70% for pure $\mathcal{Q}$-learning. Maire (2000) also found that advantage learning improved performance.

To further analyse the performance of the WFNN algorithm a more accurate evaluation method is required. Simply monitoring how many simulated boats reach their target does not reveal the quality of the controller. Many of the simulated boats glance past the target rather than stopping at the target. A measure of the directness of approach trajectory and ability to hold station is also required.

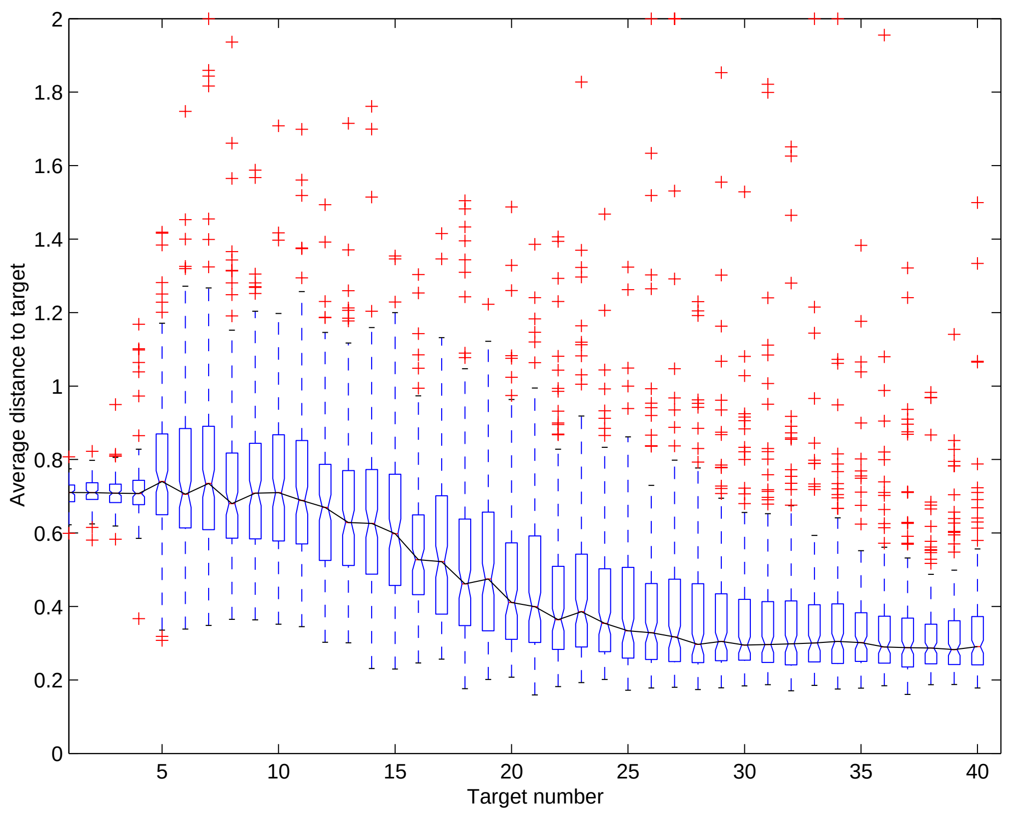
A more accurate evaluation task would be to place targets one unit away from the simulated boat in a random direction and allow 200 time steps for the boat to approach the target and maintain station. The evaluation measure is the average distance from the target over the 200 steps. The number of targets was increased from 10 to 40 to provide evidence of convergence. Performance is expected to improve on each trial. Recording average distance rather than just the ability to reach the target ensures that controllers that fail to hold station don't receive a high rating. Random firing of thrusters achieved an average distance of 1 unit (no progress) while a hand coded controller could achieve 0.25. The learning algorithm reduced the average distance with time, eventually approaching the performance of the hand coded controller.

In addition to a more effective evaluation task, a better statistical presentation and evaluation of the results is necessary. Giles and Lawrence (1997) argued against analysis based only on the mean performance or the mean and standard deviation. Analysis of the mean and standard deviation is only appropriate when the results follow a Gaussian distribution. This is not the case in many machine learning experiments. Our research follows the suggestions of Giles and Lawrence by using *box and whisker* plots to present the results of the experiments.

Box and whisker plots comparing 140 controllers trained with Q-learning and advantage learning are shown in Figure 4.2. The graphs confirm that the distribution of the results is not Gaussian. The median distance to the target is the horizontal line in the middle of the box. The notch about the median represents the 95% confidence interval for the median. The upper and lower bounds of the box show where 25% of the data lies above and below the median; therefore, the box contains the middle 50% of the data. Outliers are represented by a '+' sign. Outliers are defined as data points that deviate from the median by more than 1.5 times the range between the upper and lower ends of the box (the interquartile range). The whiskers show the range of the data, excluding the outliers.

The box and whisker plots show that advantage learning converges to good performance more quickly and reliably than Q-learning and with fewer and smaller magnitude spurious actions. Gradual improvement is still taking place at the 40th target. The outliers on the graph for Q-learning show that the policy continues to produce erratic behaviour in about 10% of cases.

The state-action deviation problem was revealed through viewing the movement of



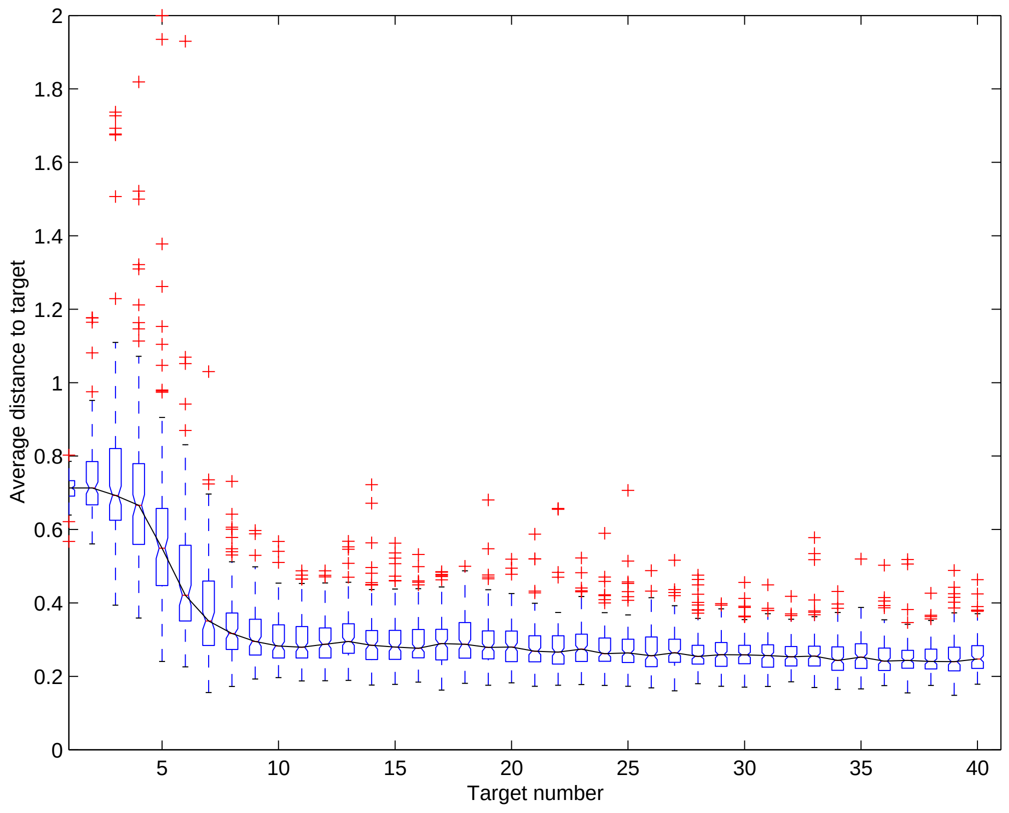


Figure 4.2: Performance of 140 learning controllers using Q-learning (*upper graph*) and advantage learning (*lower graph*) that attempt to reach 40 targets each placed one distance unit away. Outliers are clipped to a distance of 2.

the action-values, as in Figure 4.1. Observing these movements also revealed a tendency for the action-values to rise over time, causing a degradation in performance.

4.3 Rising $\mathcal{Q}$

Thrun and Schwartz (1993) observed a fundamental flaw in $\mathcal{Q}$-learning with function approximation. The $\mathcal{Q}$-learning update uses the value of the highest valued action in the next-state, $\max_{u_{t+1}} \mathcal{Q}(x_{t+1}, u_{t+1})$. If the action-values are represented approximately they can be regarded as being stored with noise. Using the maximum of several noisy action-values is an overestimate as,

$$E[\max(\vec{a})] \geq \max(E[\vec{a}])$$

where $\vec{a}$ is a vector and E is the expectation operator. In $\mathcal{Q}$-learning's case,

$$E\left[\max_{u_{t+1}} \mathcal{Q}(x_{t+1}, u_{t+1})\right] \geq \max_{u_{t+1}} E[\mathcal{Q}(x_{t+1}, u_{t+1})]$$

The systematic overestimation causes the action-values to rise over time as the $\mathcal{Q}$-learning update repeatedly overestimates the value of the next-state.

All continuous state tasks described in this thesis suffer from the rising $\mathcal{Q}$ problem. To ensure that the WFNN algorithm or the particular experimental tasks selected were not solely responsible for the problem, experiments were carried out using Lin's (1992) QCON architecture on a range of simple reinforcement learning tasks. The rising $\mathcal{Q}$ problem occurred in every instance. The simple remedy of making the reward more negative just causes the values to start lower, they still rise. Trying to regulate the action-values downward can prevent $\mathcal{Q}$ from rising; however, little learning occurs.

The overestimation phenomenon is caused by error in the represented action-values. Any measures that help to reduce erroneous data reaching the learning system, or reduce errors in representing the action-values, will help to reduce the rising $\mathcal{Q}$ problem. Errors in the state representation and reward function caused the action-values to rise quickly in several of the experiments. This suggests that the first step after observing a rise in $\mathcal{Q}$ should be to perform sanity checks on data and re-think the reward function. Measures that improve convergence in general reduce the rising $\mathcal{Q}$ problem by reducing the errors in representing the action-values (Thrun and Schwartz, 1993).

The problem can be addressed more directly for some specialised approximators. Gordon (1999) defined a class of approximators that do not suffer from overestimation;

however the approximators in this class may not be sufficiently powerful to describe the action-values. Ormoneit and Sen (1999) described an estimate of the overestimation for radial basis function approximators. Smart and Kaelbling (2000) approached the rising $\mathcal{Q}$ problem by attempting to avoid extrapolation. The approach is sensible since extrapolated action-values are less reliable than interpolated action-values. Smart and Kaelbling's method tried to bound the reliable region of the action-values with a convex hull. They noted that their method's conservatism could limit generalisation; however, they found it was not a serious problem for their trial tasks. Unfortunately, this method is not directly applicable to the WFNN algorithm.

4.3.1 A Partial Solution through Advantage Learning

Advantage learning addresses the problem of rising $\mathcal{Q}$ in two ways. Firstly, advantage learning appears to ease the load on the function approximator and makes its estimates more accurate. It reduces the probability of overlapping error bounds in value estimates. Secondly, we found that it is possible to eliminate the systematic overestimation by judiciously setting the value of k, the advantage learning expansion factor.

In advantage learning, the maximum is taken twice: once for the current state and once for the next-state (see Equation (4.1)). Under the assumption that the overestimation of value will on average be the same for the state and the next-state, k can be set to subtract one overestimation from the other.

If Y is the overestimation of the maximum, and $V(x)$ is the true value of a state, then the overestimation in the update equation is,

$$\begin{aligned}\text{overestimation} &= \left[\tfrac{1}{k}\left[r\left(x_t,u_t,x_{t+1}\right)+\gamma\left(V\left(x_{t+1}\right)+Y\right)\right]+\left(1-\tfrac{1}{k}\right)\left(V\left(x_t\right)+Y\right)\right]\\ &\quad-\left[\tfrac{1}{k}\left[r\left(x_t,u_t,x_{t+1}\right)+\gamma V\left(x_{t+1}\right)\right]+\left(1-\tfrac{1}{k}\right)V\left(x_t\right)\right]\\ &= \tfrac{1}{k}\gamma Y+\left(1-\tfrac{1}{k}\right)Y\\ &= \left(1-\tfrac{1+\gamma}{k}\right)Y\end{aligned}$$

If $k = 1-\gamma$ the overestimation should be eliminated. A disadvantage of the approach is that the gains on the individual overestimations rise as k decreases, thereby increasing the variance. The effect of modifying k and other parameters is analysed quantitatively in following sections.

4.4 Parameter Sensitivity

The WFNN algorithm has a number of design parameters. If accurate setting of the design parameters is important and there is no procedure for choosing the parameters then using the algorithm would be impractical.

The major design parameters are as follows:

1. k, the advantage learning factor, and γ, the reinforcement learning discount factor;
2. the neural network training rule;
3. α, the neural network learning rate;
4. the number of neural network hidden neurons;
5. the number of wires;
6. the number of experience replays processed by the learning system between performing actions—called here the *thinks* per action;
7. c, the smoothing factor; and
8. the approach to the error in the derivative calculation.

The importance of the design parameters was evaluated using the simulation task. The performance measure was the average distance from the target position over 200 steps. Box and whisker plots are used to visualise performance. A single box and whisker summarises performance over all targets from 1 to 40, thus it indicates a mixture of learning speed, stability and controller efficiency.

In order to objectively compare performance nonparametric (distribution free) statistical tests were applied. Unlike parametric statistical tests, these tests do not assume that the data has a particular distribution. The Wilcoxson rank sum test gives the probability that two sets of samples are from data with the same median; the Kolmogorov-Smirnov test gives the probability that the samples are from data with the same distribution (Conover, 1999).

Twenty simulation runs were made for each combination of parameter settings. Factors γ and k are analysed together since they are dependent. The default parameter settings are $k = 0.2$; $\gamma = 0.8$; $\alpha = 0.25$; neural network hidden neurons=10; wires=9; thinks per action=200; $c = 0.01$; and calculation of derivatives with the approximate equations. As shown in following results, these settings give acceptable performance.

4.4.1 The Advantage Learning Factor and Discount Factor

The discount factor, γ, weights the importance of future rewards against current rewards. γ must be close to 1 in order to solve problems over a long time horizon.

Figure 4.3 compares median performance of the WFNN algorithm with various settings of γ and k. The data is from 3000 training runs, each run consisting of 40 targets. Darker colour represents better performance. Outliers are shown as dots, and clipped to 1.3 distance units. The graph shows that performance is poor when γ is below around 0.6 independent of the value of k. The line along the top of the graph shows the mediocre performance of $\mathcal{Q}$-learning, where $k = 1$. Best performance occurs when $\gamma = 0.8$ and $k = 0.2$. The result supports the theoretical prediction that $k = 1 - \gamma$ eliminates the rising $\mathcal{Q}$ problem.

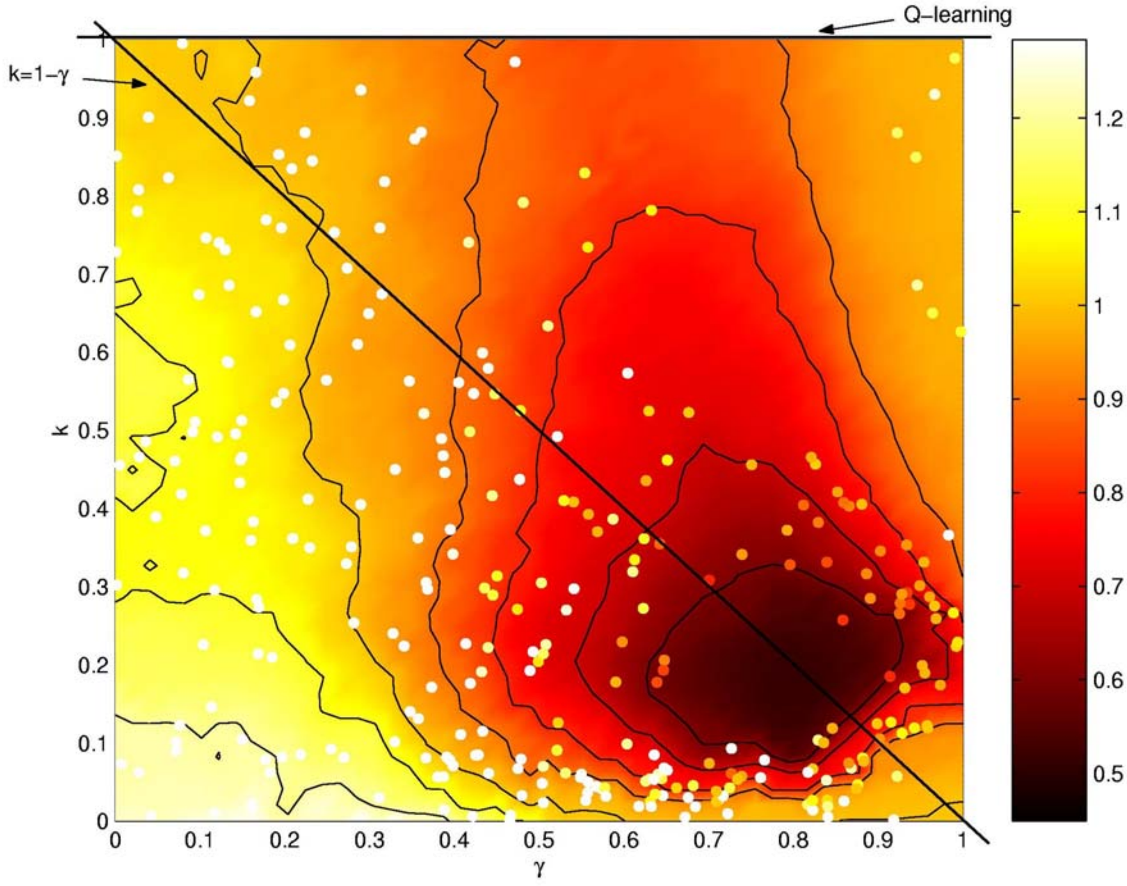


Figure 4.3: Median performance of the WFNN algorithm for the boat navigation task. Darker colours represent better performance.

Figure 4.4 shows the lower quartile performance. This represents near best case performance: 25% of controllers can be expected to exceed this performance level. The graph shows that best case performance can be high for high values of γ and k near 0; however, Figure 4.3 shows that median performance is poor.

Figure 4.5 explains the poor median performance with small k. The graph shows the interquartile range, a distribution free measure of variation. The interquartile range is high when k is near 0, causing inconsistent performance. Based on the results of these experiments the parameters were set to $\gamma = 0.8$ and $k = 0.2$ for the experiments reported in later chapters. The setting of γ is too low for *pure* delayed reward problems, but seems adequate for delayed reward problems.

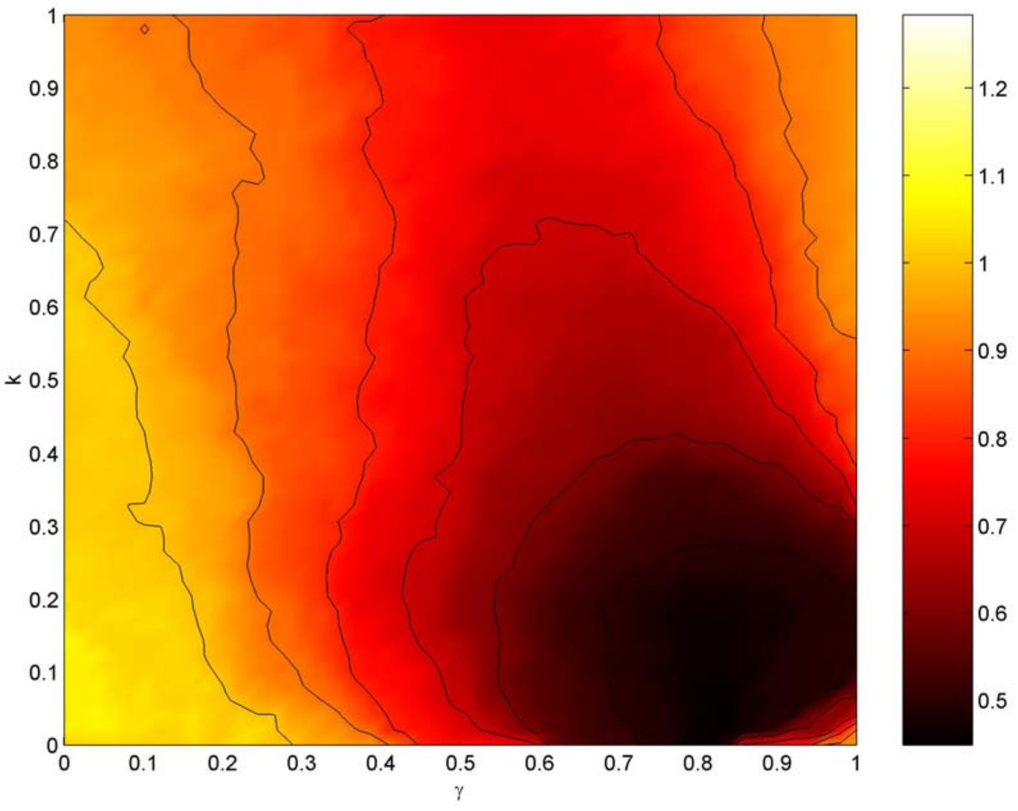


Figure 4.4: Lower quartile (near best case) performance

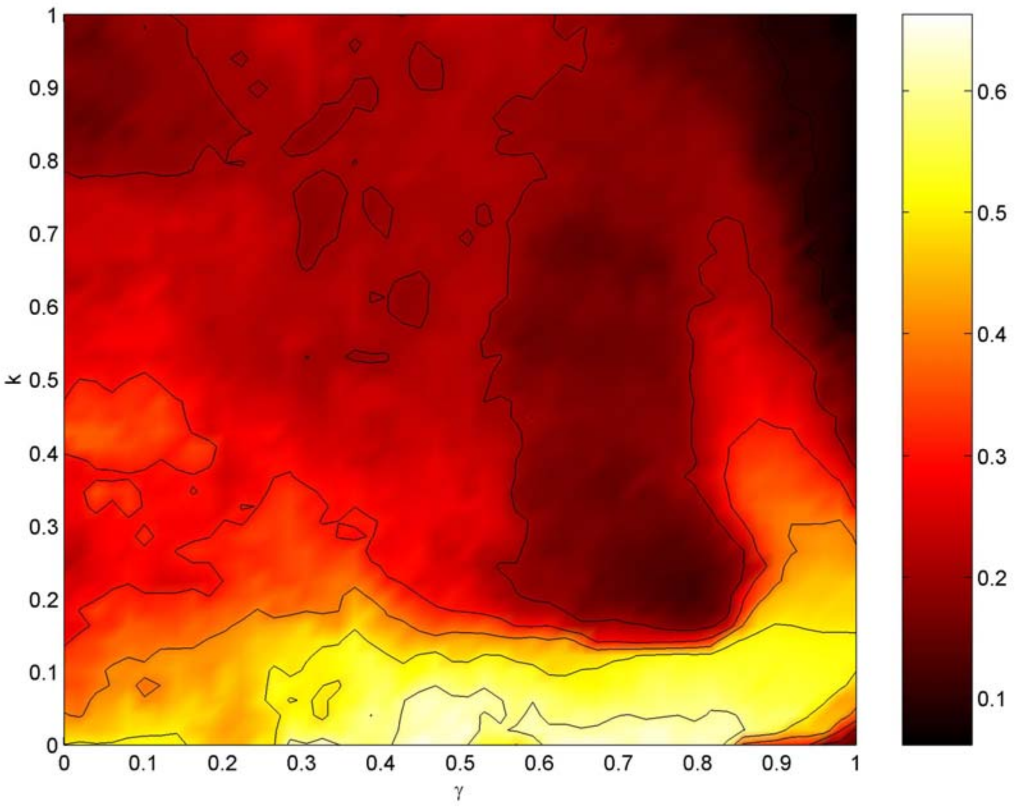


Figure 4.5: Interquartile range; light colours indicate inconsistent performance

4.4.2 Neural Network Training Rule

The experimental implementation of the WFNN algorithm used incremental backpropagation to train a feedforward neural network, or multilayer perceptron (Rumelhart et al., 1986). In incremental backpropagation the neural network weights are updated after every training example based on the error gradient for that example. This form of stochastic gradient descent is the simplest approach to neural network training.

There are a number of more sophisticated, 2nd order neural network training algorithms. These algorithms could be expected to converge faster than the simple incremental algorithm. The algorithms are based on the assumption that there is a fixed correct output to match the input for the neural network. This is reasonable for pattern classification tasks or copying existing behaviours. However, for reinforcement learning it is not appropriate because of the recursive nature of the updates. For the WFNN algorithm the neural network training error is small during learning—even when learning fails and the system diverges. There is little evidence that 2nd order algorithms can find better neural network weights given the same amount of data (Lawrence and Giles, 2000; Gorse et al., 1997).

We found that more complicated neural network training algorithms, batch-based training with and without momentum (applying a low pass filter to weight changes), had no significant effect on the performance of the WFNN algorithm.

4.4.3 Learning Rate

The learning rate, α, strongly influences performance on standard supervised learning problems (Plaut et al., 1986). A learning rate that is too small leads to slow convergence and the possibility of being stuck in local minima. High learning rates can prevent convergence because the system repeatedly steps over useful minima.

The best learning rate can be different for each layer of the neural network. Monitoring the progression of weight values graphically during our simulation experiments revealed a significant imbalance between the magnitude of weights to the hidden layer and the magnitude of weights to the output layer. The imbalance varied with the number of inputs and the number of hidden neurons. As suggested by Plaut et al. (1986), and justified by Schraudolph and Sejnowski (1996), the relationship between the learning rates of the different layers of the neural network had a strong influence over performance. When the learning rates of the input and hidden layers were the same but the number of input nodes and hidden nodes were different, the magnitudes of weights from the nodes is unbalanced and learning performance is poor. Following the $\sqrt{\text{fan-in}}$ heuristic given by Plaut et al. (1986) resulted in balanced weight magnitudes between layers and higher performance. The *fan-in* is the number of input weights into a node. The learning rate for each layer becomes the global learning rate divided by the $\sqrt{\text{fan-in}}$.

Figure 4.6 shows the performance with various global learning rates. The relationship between learning rate and performance appears to be the same as for standard supervised learning. A small learning rate resulted in slow learning; the performance was barely superior to the initial random weights. High learning rates result in fast learning with high median performance. The outliers show that the risk of divergence increased, however. The sensitivity to the learning rate setting was no worse than for standard supervised learning; a wide range of learning rates result in good performance; hence it is practical to find a suitable setting.

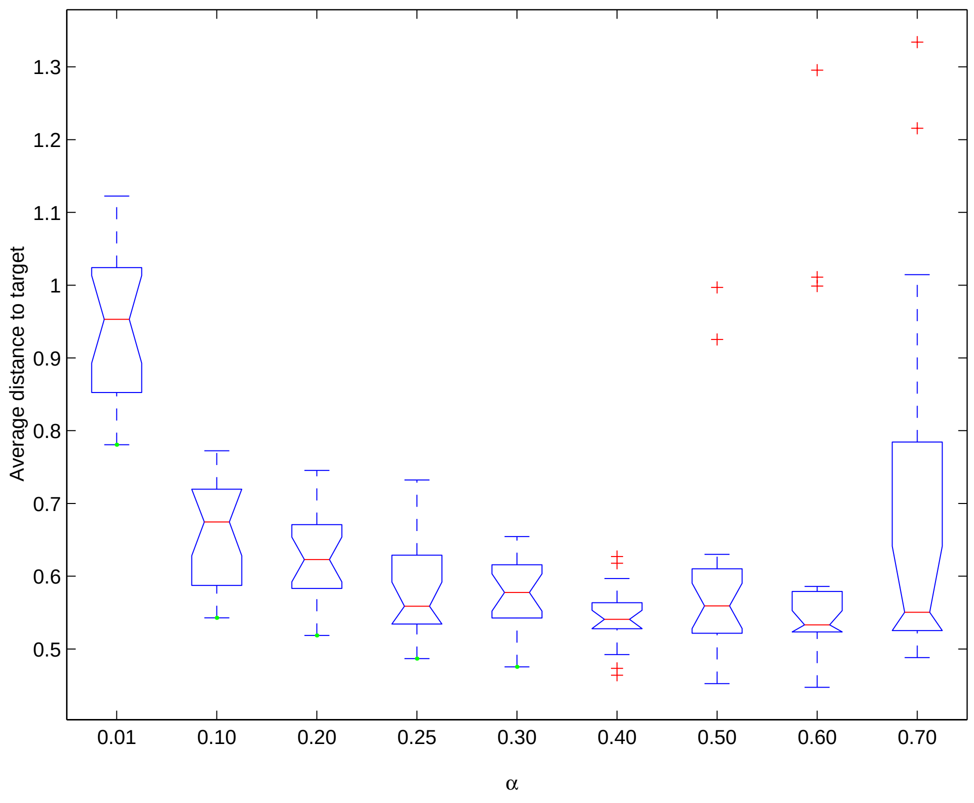


Figure 4.6: Performance versus learning rate

4.4.4 Number of Hidden Neurons

The number of hidden neurons in a neural network determines range of functions that can be represented. The minimum number of hidden neurons required varies with the complexity of the learning problem, including the number of state and action variables.

The experimental results shown in Figure 4.7 show that the WFNN algorithm successfully learnt the boat navigation task provided there were at least 3 hidden neurons. Using more than the minimum number of hidden neurons had no deleterious effect apart from increasing processing requirements. The learning time in terms of the number of experiences processed was unaffected. These results are in accordance with the general finding that excess hidden neurons cause few problems (Lawrence and Giles, 2000). As far as processing resources allow, it is safer to choose a high number of hidden neurons.

4.4.5 Number of Wires

The number of wires combined with the number of hidden neurons determines the range of value functions that can be represented by the WFNN learning system. The

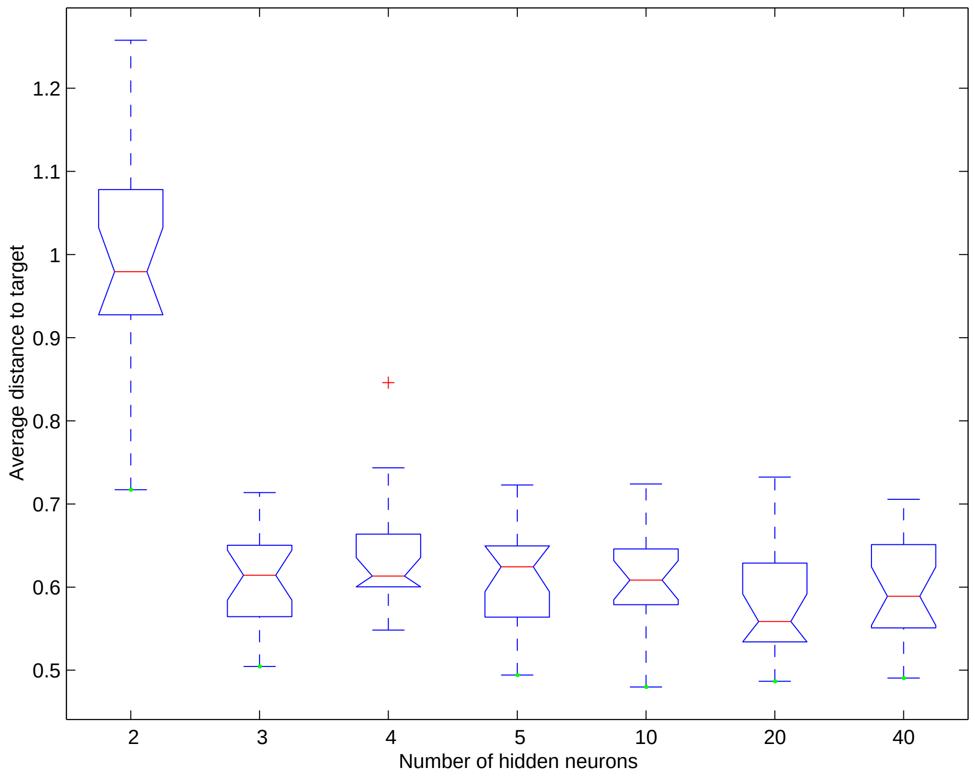


Figure 4.7: Performance versus number of hidden neurons

experimental results shown in Figure 4.8 are analogous to those for the hidden neurons. A sufficient number of wires is required. With only one wire the learning system can only represent the case in which all actions have the same value; hence performance was no better than random. Reasonable performance began with two or three wires. Performance was best with five or more wires. Analogous to the hidden neurons case, excess wires increased processing requirements but did not increase learning time in terms of the number of experiences processed. As far as processing resources allow, it is safer to choose a high number of wires.

4.4.6 Thinks per Action

As experiences are replayed over and over again it raises the question of how many *think* cycles should be carried out per action cycle. Figure 4.9 shows performance versus the number of *thinks*. Performance improved as the number of thinks increased but declined dramatically for high numbers of thinks, suggesting an overfitting problem. In our robot experiments an upper limit of around 500 thinks was imposed by real-time constraints so overfitting due to over-thinking was unlikely. However, if processing was performed off-line, without real-time constraints, then overfitting could be a significant problem.

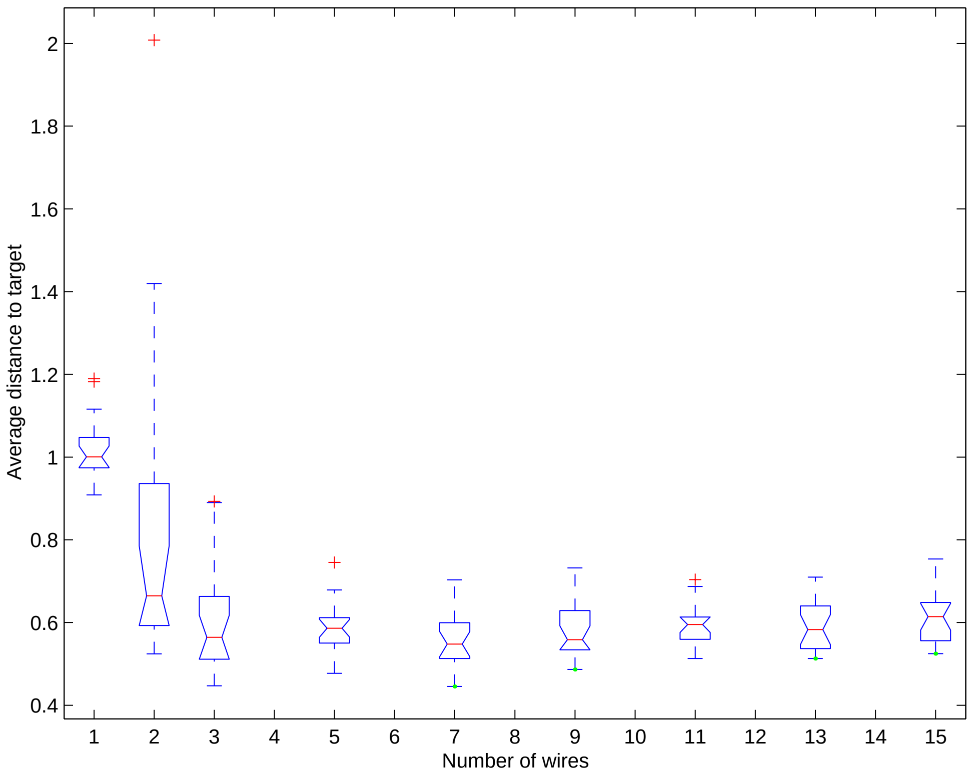


Figure 4.8: Performance versus number of wires

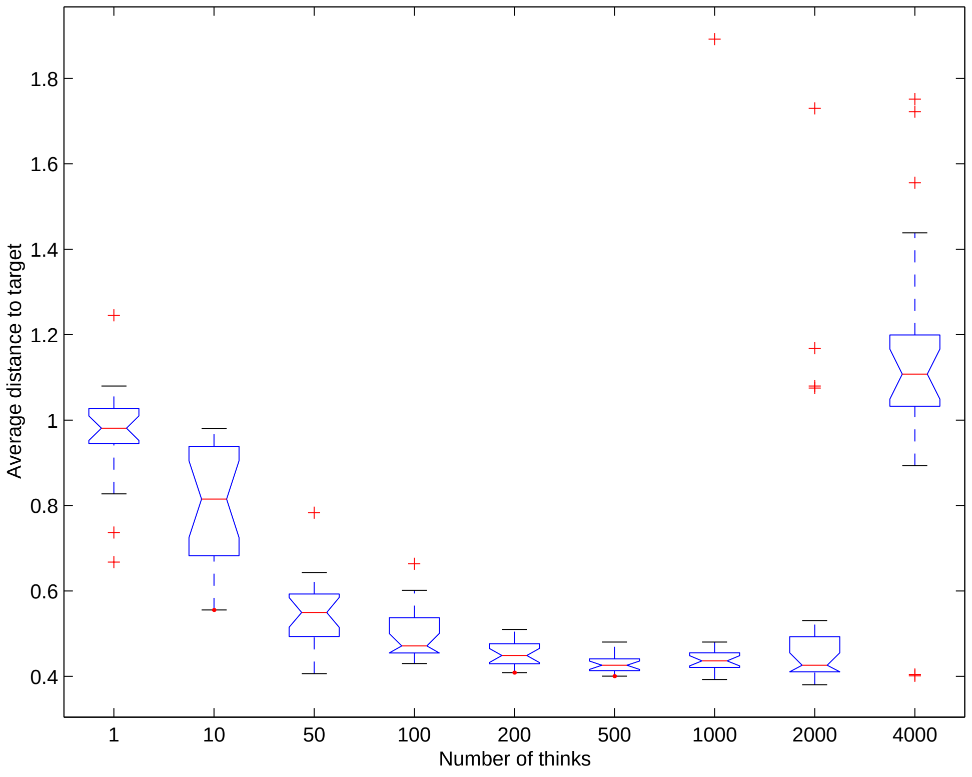


Figure 4.9: Performance versus number of thinks

4.4.7 Smoothing Factor

The effect of the smoothing factor, c, in Equation (3.1), was experimental evaluated. The results are shown in Figure 4.10. There were no significant differences in performance for different values of c, except that performance was poor when c equals zero, indicating that a smoothing factor is essential.

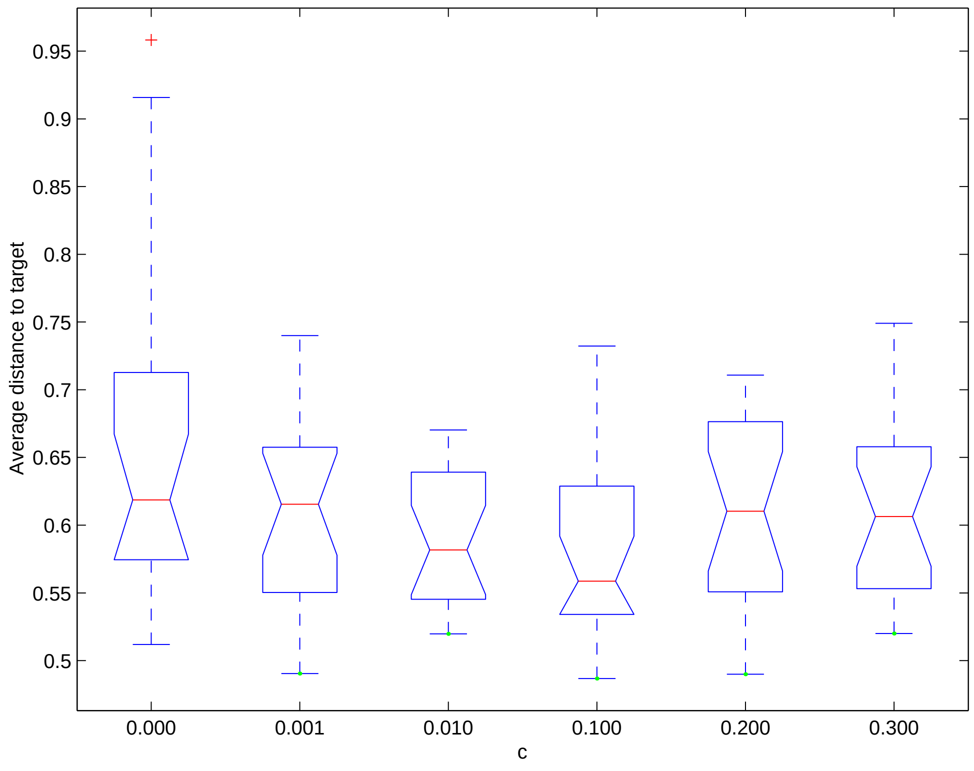


Figure 4.10: Performance versus smoothing factor

4.4.8 Derivative Error

The derivative calculation in Equations (3.4) and (3.5) is inexact for the wire with the highest Q. The error can be approached by:

1. ignoring the error and using the approximate equations;
2. calculating the derivatives using finite differences instead of the approximate equations; or
3. calculating the derivatives for the wire with the highest Q using finite differences, and using the approximate equations for the other wires.

Evaluating these three methods revealed no significant difference in performance. The first approach is preferable because the inexact equations require less processing than calculation using finite differences.

4.5 Pure-Delayed Reward: The Cart-Pole Problem

The boat navigation task is a *delayed reward* problem: a reward is available at every time step, but it is not a complete, immediate, evaluation of the value of an action in a state. Actions generated from the control system need to take the future into account. For example, in the boat navigation task, the policy that brings the highest immediate reward would be to always travel towards the target at maximum velocity. However, travelling at maximum velocity leads to travelling past the target as the boat has momentum. In the long-term, it is more valuable to lower the velocity as the boat nears the target. At first the learning system does tend to send the boat past the target, but it learns to control speed appropriately. It is clear that the learning system is solving a delayed reward problem.

The boat navigation task is not a *pure-delayed reward* problem. In pure-delayed reward problems the reward is constant until the end of the trial, when a reward reflecting success or failure of the entire trial may be given. In the next section the WFNN learning system is applied to a pure-delayed reward problem in simulation.

4.5.1 The Learning Task

The task in the cart-pole problem is to balance a pole by applying forces to a cart running on a track of fixed length. The configuration is shown in Figure 4.11 (from software by Anderson, 1998).

The dynamics of the cart-pole system are [1] :

$$\begin{aligned}\ddot{x} &= \frac{F + m_p l\left[\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta\right] - \mu_c \operatorname{sgn}\dot{x}}{m_c + m_p} \\ \ddot{\theta} &= \frac{g\sin\theta + \cos\theta\left[\frac{-F - m_p l\dot{\theta}^2\sin\theta + \mu_c \operatorname{sgn}\dot{x}}{m_c + m_p} - \frac{\mu_p\dot{\theta}}{m_p l}\right]}{l\left[\frac{4}{3} - \frac{m_p\cos^2\theta}{m_c + m_p}\right]}\end{aligned} \tag{4.3}$$

[1]The variables in Equation (4.3) are F N, the force applied to the cart (the action); θ radians, the angle of the pole; and x m, the position of the cart. The constants are $g = 9.81\text{m/s}^2$, the acceleration due to gravity; $m_c = 1.0$kg, the mass of the cart; $m_p = 0.1$kg, the mass of the pole; $l = 0.5$m, half of the pole length; $\mu_c = 0.0005$, the friction coefficient of the cart on the track; and $\mu_p = 0.000002$, the friction coefficient of the pole on the cart. Position is calculated using Euler integration with a time step of 0.02 seconds.

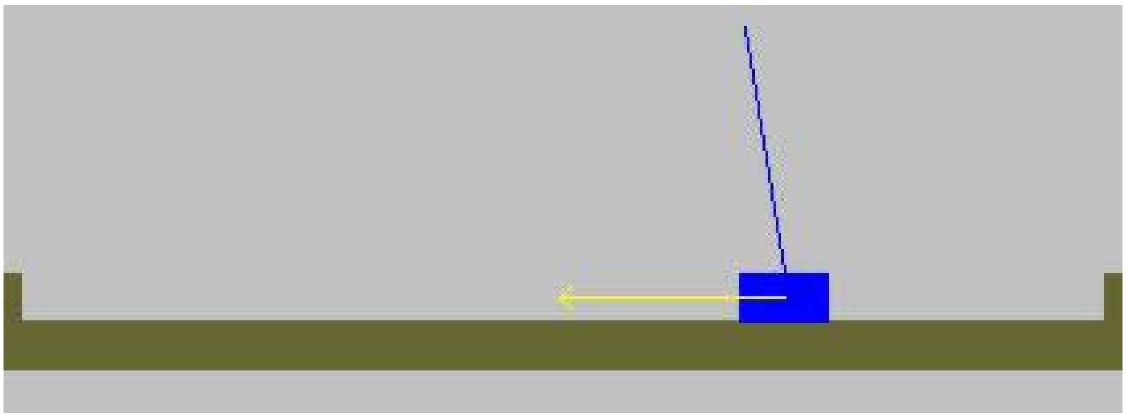


Figure 4.11: Pole balancing. Forces are applied to the cart to keep the pole upright. The cart must remain within the bounds of the track.

The cart-pole problem was solved using trial and error by a learning system developed by Michie and Chambers (1968). Their BOXES system was an early reinforcement learning system that used a discrete representation of both states and actions. More recently, the problem was used by Barto et al. (1983) as a test for Actor-Critic learning. In their experiment the actions were set to +10 or -10 newtons and the state space was also discretised. The reward was zero until the pole falls past 12 degrees or hits the end of the track, which was 2.4m long. The state representation in this experiment was the scaled position and velocity of the cart and the scaled angle and angular velocity of the pole.

Baird and Klopf (1993) applied the memory-based wire fitting approach to the task with continuous states and actions (-10N — +10N). The system was able to solve the cart pole problem only if negative reward proportional to the square of the pole angle was supplied and the length of the track was infinite.

4.5.2 WFNN Results

The WFNN system learned to control the cart-pole system with continuous states and actions, pure-delayed reward, and a finite track. On some runs it learnt to balance the pole for 400,000 steps after only 100 attempts, learning at a similar rate to the Actor-Critic system (Barto et al., 1983). However, the WFNN controller cannot be considered successful since the policy degraded with time.

Figure 4.12 shows that the performance of the controller increased from random (less than 50 steps before failure) to mediocre (a little better than 100 steps) after 20 attempts. At 120 attempts the pole balanced for 400,000 steps. Following the success, performance degraded quickly back to mediocre. At 170 attempts performance again improved and some consistency is achieved over the next 50 attempts. Performance then degraded back to mediocre and continued to degrade through at least the next 150 attempts.

The typical progression of controller development was as follows:

1. Randomness: the cart stays near the middle of the track; the pole falls within a few steps.
2. Single directional movement: the cart moves toward the end of the track in the direction the pole is leaning. Failure occurs when the cart collides with the end of the track.
3. Single lap: the cart moves toward the end of the track in the direction the pole is leaning; however the cart moves quickly enough to make the pole lean the other way. The cart then moves in the new lean direction and collides with the other end of the track.
4. Multiple laps: as in single laps, except that the cart changes direction more than once. As learning progresses the cart makes more laps and stays further away from the end of the track.
5. Near stationary balancing: the cart stays near the middle of the track, balancing the pole for many steps.
6. Degradation: behaviour reverts to pattern 1, 2, or 3.

4.5.3 Discussion

Convergence to a good policy is not guaranteed, but once a good policy has been found it seems reasonable to expect that it will not be lost. Inconveniently, the intuition is incorrect. In each time step the learning update rule changes the neural network weights. The changes are dependent on data entering the learning system and the current state of the learning system.

The nature of the data entering the learning system can change when the controller begins to perform well. For example, a mediocre controller for the cart-pole system receives feedback, in the form of a negative reward, whenever it fails to balance the pole. If the controller improves so that it balances the pole for a long time it ceases to receive feedback. The learning system continues to perform its learning update; however, it is bootstrapping, updating based on its own internal estimates, rather than learning from data entering the system. Even though we can reuse experience of failure from previous

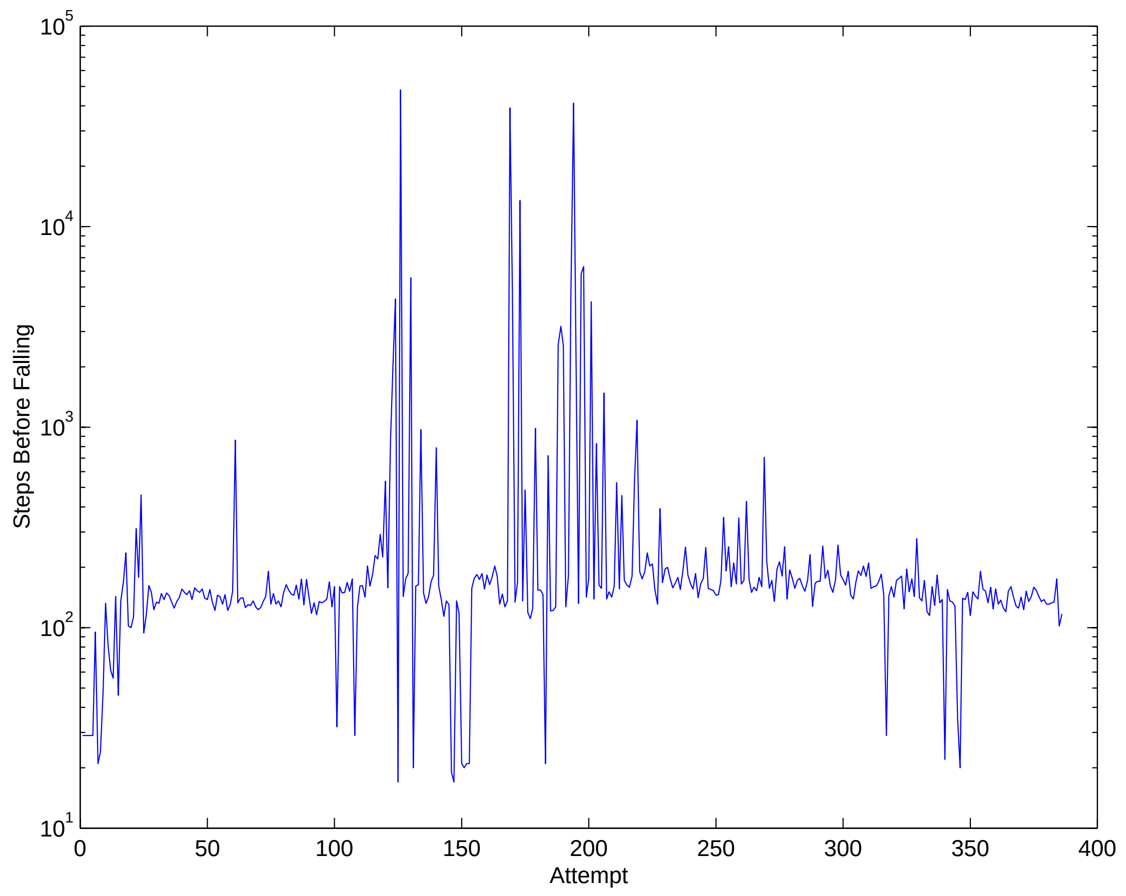


Figure 4.12: After learning to balance the pole the policy degrades.

controllers, through *experience replay*, the system has become closed, and the distribution of experiences changes when the controller is successful. Excessive bootstrapping and reuse of data leads to unlearning.

We tried several approaches in an attempt to reduce policy degradation:

- disposing of experiences after a time limit or after they have been replayed several times to try to reduce over-training;
- maintaining separate databases for reward = 0 and reward = -1 to try to maintain a balance of these experiences;
- adjusting the number of hidden nodes;
- using various fixed learning rates; and
- regularising the neural network weights.

These approaches were unsuccessful, either having no effect, preventing learning entirely, or delaying degradation only by slowing learning.

Stopping learning

Policy degradation problems could be avoided if the system was configured to stop learning when a successful controller has been found. Once learning stops the controller would inconveniently be unable to adapt to new circumstances. However, the main problem is detecting when to cease learning, or how to detect when the controller is performing well. Like a poker player, you've got to know when to hold 'em; know when to fold 'em; know when to walk away; and know when to run (Rogers, 1978).

A performance target can be set using domain knowledge, for example balancing the pole for 10,000 steps. When the performance goal is met learning can be turned off. The approach is flawed and it failed when we applied it to the WFNN controller for the cart-pole task.

When excellent performance is detected the controller may have already degraded. In this task, reaching the performance target usually occurs when the pole has been balancing in one position on the track for many time steps. The result is over-training about balancing in that position and consequent loss of the ability to balance the pole in other positions.

If balancing in one position leads to over-fitting, then cancelling the trial and resetting the cart-pole system after, for example, 200 steps seems to be a sensible approach. The performance target is to balance the pole successfully for several consecutive 200 step trials. This approach could be expected to prevent over-training by spreading learning efforts over the space of starting positions. Again, the approach is flawed and failed.

The system learnt a policy that balances the pole for slightly more than 200 steps. Accordingly, if the number of steps per trial was n, then the learned policy balanced the pole for slightly more than n steps. This type of policy arises through the typical progression of development in solving the problem: randomness, single directional movement, single lap, multiple laps, near stationary balancing, followed by degradation. Limiting the task to n steps stalled development of the controller in the multiple lap stage, when it was capable of performing enough laps to consistently reach n steps.

Slowing learning

Rather than trying to stop the learning process at the right time, a less abrupt approach is to reduce the learning rate to zero gradually. Strictly, convergence requires a progression rather than a fixed learning rate (Watkins and Dayan, 1992). The problems are similar to those of stopping learning: the controller's ability to react to changes is compromised, and a learning rate reductions schedule must be chosen.

Although a suitable learning rate reduction schedule might be found by trial and error, the schedule is *task-dependent*. In simulation studies it might be acceptable, or even unnoticed; however, an indivinable, task-dependent parameter is unacceptable for robotics problems. If this type of parameter is set by hand, through multiple tests, for each robotic task, then the task has shifted from robot learning to human perseverance. In contrast to this task-dependent drudgery, the parameters and rules for choosing parameters described in this chapter are exactly those used in the robotics experiments in Chapters 6 and 7.

The nature of pure-delayed reward tasks

The learning system's tendency to degrade when applied to the cart-pole problem, yet remain stable for the boat navigation task, is due to the reward structure: when cart-pole controller performance is good the reward is constant and provides no feedback.

Other common pure-delayed reward reinforcement learning problems involve finishing a task as soon as possible in order to receive a terminating positive reward, or cease receiving a constant negative reward. The first completion of the task is expected to occur eventually due to the random actions generated by the controller. However, most continuous action learning systems initially generate small magnitude actions, and no sequence of small magnitude actions can complete the task. Pure-delayed reward test tasks are designed to demonstrate the abilities of discrete action controllers; consequently, they do not require continuously variable actions. Furthermore, continuously variable actions can be a hindrance to solving such problems.

Robotics tasks that require continuous actions often involve delayed rewards; however, they rarely involve pure-delayed rewards. Continuous action tasks generally allow feedback during the task: measures of progression, smooth control, and energy use. Although the instability of the WFNN algorithm when applied to pure-delayed tasks is disappointing, it does not imply that the WFNN algorithm is useless for robotics tasks. Rather, it emphasises the importance of designing the task representation, including the reward function, to provide as much feedback as possible.

4.6 Summary

The simulations showed that parameters of the WFNN system can be chosen with similar ease to those for a neural network trained through incremental backpropagation. A method of choosing the advantage learning factor, k, in order to avoid the rising Q problem was proposed and evaluated successfully.

A pure-delayed reward learning task was attempted and solved quickly, but performance degraded. Attempts to prevent the degradation were unsuccessful. It is not a serious deficiency since continuous action robotics tasks rarely involve pure-delayed rewards.

The focus of our research is to investigate performance on robots. To make this step, real-time and systems integration issues must be addressed. These issues will be discussed in the following chapter.

Chapter 5

Implementation

A monk asked, "It's not clear to me, what is a 'temporary expediency' to be called?"

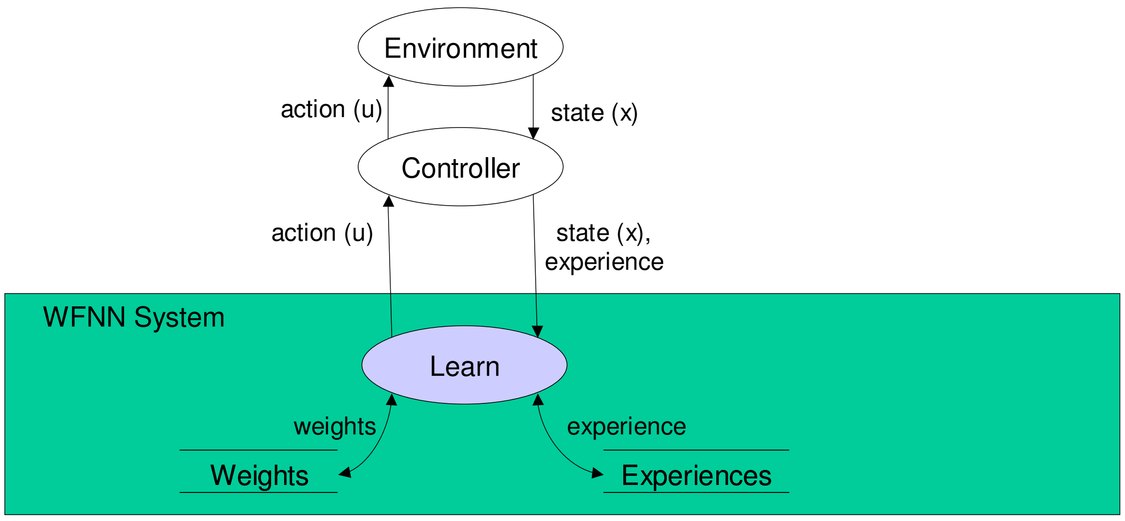
The master said, "A 'temporary expediency'."

Joshu, c. 890, no. 425

Robotics is a challenging application for reinforcement learning algorithms. Complete state data is not available, and the incomplete state data is not reliable, arrives in real-time, and is quickly out of date. Responses must be made in real-time with imperfect actuators, which do not operate instantaneously, can damage the robot, and cannot reset the system to its starting state. It is difficult to ensure safety for both people and robots when an algorithm can act unpredictably. If the system architecture does not address these issues it will be limited to operation on small robots operating in constructed, static environments. This chapter discusses architectural ideas and implementation details that contribute toward the usefulness of reinforcement learning algorithms for robotics. A series of architectures are presented, showing development from a simple, monolithic system to a scalable, real-time approach.

5.1 Managing Unreliable Sensor Data

Control systems generate actions based on inputs from sensing systems. The closed-loop nature of a control system helps to compensate for any transient errors in the sensing system. In learning systems the situation is less desirable: transient sensing errors can easily misdirect a learning controller. The learning system has no common sense that allows it to separate reasonable sensor readings from the ridiculous. Therefore, the learning system must be protected from erroneous data through sanity checking and filtering. If measured data is believed to be of questionable accuracy, for example if the correlations are low in a computer vision template matching system, the system should not learn from the data. State data should also be ignored by the learning system during irrelevant state transitions. An irrelevant state transition occurs when the state data changes in a way that should not be the concern of the learning system. The boat navigation task in Chapter 3 is an example of a reinforcement learning problem with occasional irrelevant state transitions. The state changes when a new target is selected, however the learning system is not expected to predict where new targets will appear; the state transition that occurs when the target is moved needs to be hidden from the learning system.

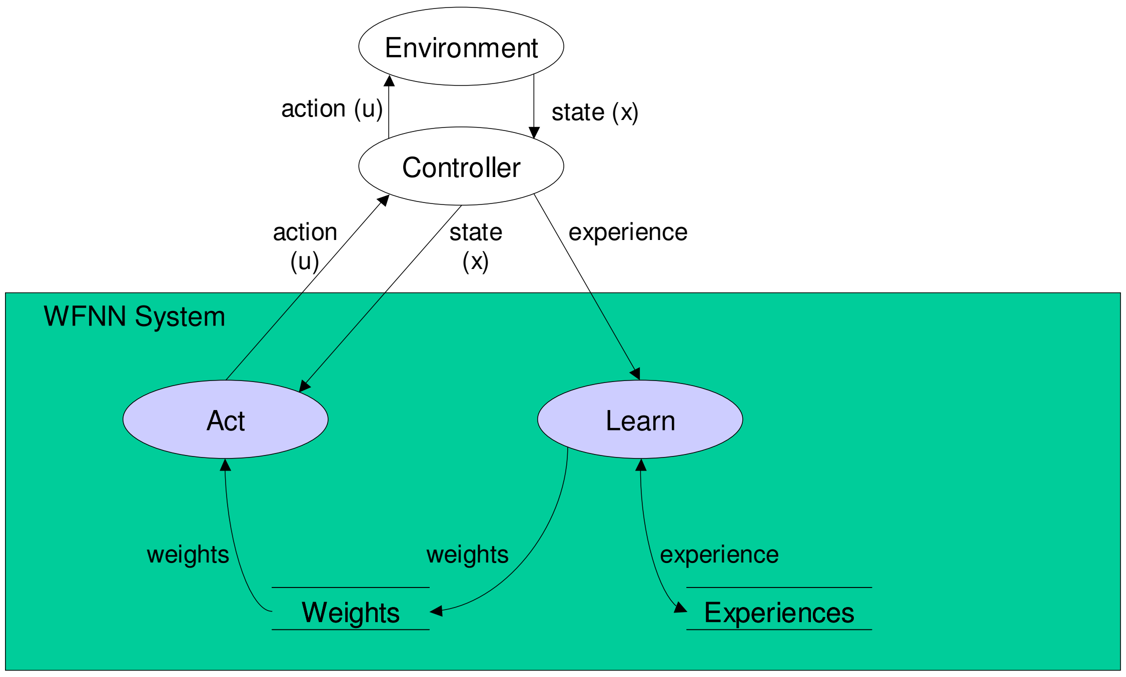


```
1   repeat {
2     measure state (at regular time interval)
3     if the state data is valid then {
4       learn(state, action, prev_state, prev_action, reward)
5       perform action
6     }
7   }
```

Figure 5.1: DFD and pseudocode example for the *learn* architecture

Figure 5.1 shows our approach that supports ignoring suspect or irrelevant state data. The approach will be referred to as the *learn* architecture as it is the only method in the interface. The pseudocode in the lower half of the figure shows how a controller uses the interface. The method `learn` returns an action suggested by the learning system and learns from the previous state transition and reward. It implements Steps 1c, 1d, 1i, and Parts 2 and 3 in their entirety, from the WFNN algorithm described in Chapter 3. The Data Flow Diagram (DFD) (DeMarco, 1978) for the *learn* architecture shows information flows as arrows; processes that transform or retransmit data as ovals; and data storage as parallel horizontal lines.

A major drawback of the single function `learn` interface is that it is not always appropriate for the learning controller to make no action when the state data is suspect: taking no action until sensing data improves could leave the robot trapped in a static situation. A more helpful approach is to execute the action suggested by the controller based on the best possible estimation of the state, but ignore the data for the purposes of learning. To support the idea the *act-learn* approach is proposed. A DFD and pseudocode example for the *act-learn* architecture is shown in Figure 5.2. The `act` method implements Steps 1c and 1d from the WFNN algorithm described in Chapter 3. The `learn` method implements Step 1i, and Parts 2 and 3 in their entirety.



```
1   repeat {
2     measure state (at regular time interval)
3     act(state, action)
4     perform action
5     if the state data is valid then
6       learn(prev_state, prev_action, state, reward)
7   }
```

Figure 5.2: DFD and pseudocode example for the *act-learn* architecture

5.2 Managing Real-Time Constraints

A robot that interacts with its environment in a sensible way requires real-time performance. Sensor data must be processed in real-time and the timing of the robots actions is as important as what the actions are. Figure 5.3 shows a timing diagram for a real-time reinforcement learning system. The time interval between encounters with the state valid condition is constant since the source of the condition is usually low-level hardware, for example a video frame-grabber, that outputs data at a regular time interval. To stay synchronised with the data the total processing time per iteration should be constant.

The time between state valid and action executed must be consistent and minimal to mitigate a perceptual aliasing problem that occurs due to the state changing before the action is executed. The aliasing problem becomes worse as the total iteration time decreases. The *act-learn* architecture described in the previous section assists in minimising the time between state valid and action executed because only essential calculations are performed between measuring the state and performing an action.

Idle time should be minimised since the time could have been spent on learning processing and other processing, but an overzealous attempt to minimise idle time could lead to an overrun of the total iteration time and cause the system to lose synchronisation with the state valid condition. The *experience replay* technique described in Chapter 2 allows some control over the learning processing time through adjusting the number of experience replays per processing loop.

The *act-learn-think* architecture enhances the *act-learn* architecture by allowing control over the number of experience replays. A DFD description and pseudocode example for the *act-learn-think* architecture are shown in Figure 5.4. In this architecture the interfaces to the `act` and `learn` methods are the same as described previously. `Act` still executes Steps 1c and 1d from the WFNN algorithm. Instead of learning from the data the `learn` method simply adds an experience to a database (step 1i). The method `think` processes the data, implementing Steps 2 and 3 in their entirety. The parameter `thinks` determines the number of experience replays. An interface to the WFNN algorithm *act-learn-think* interface suitable for graphical simulation of dynamic systems is described in Appendix A.

In the non-real-time simulation experiments presented in Chapters 3 and 4 the number of experience replays was fixed. This was appropriate since the purpose of those experiments was to compare various modifications to the algorithm. In the real-time

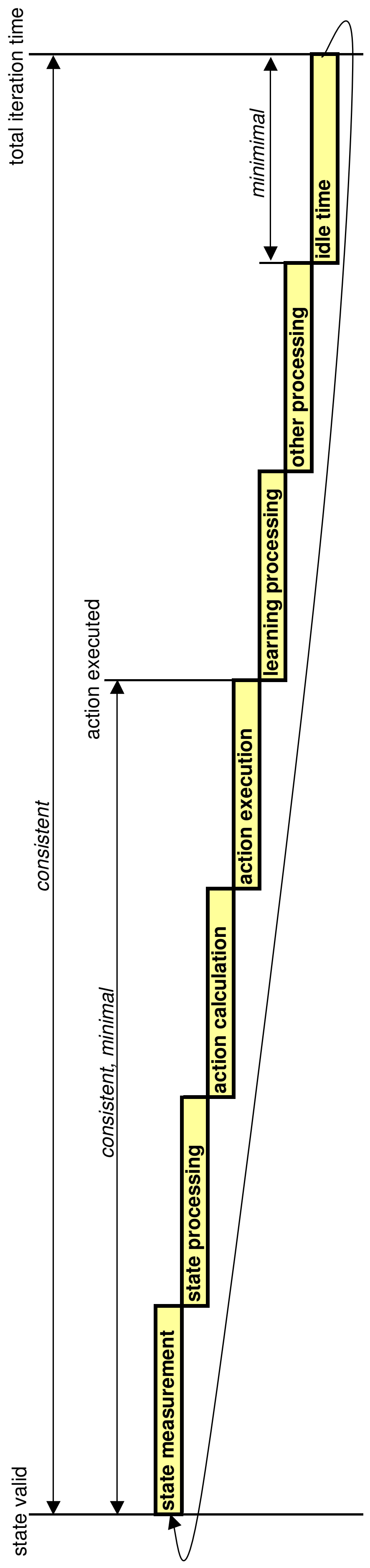
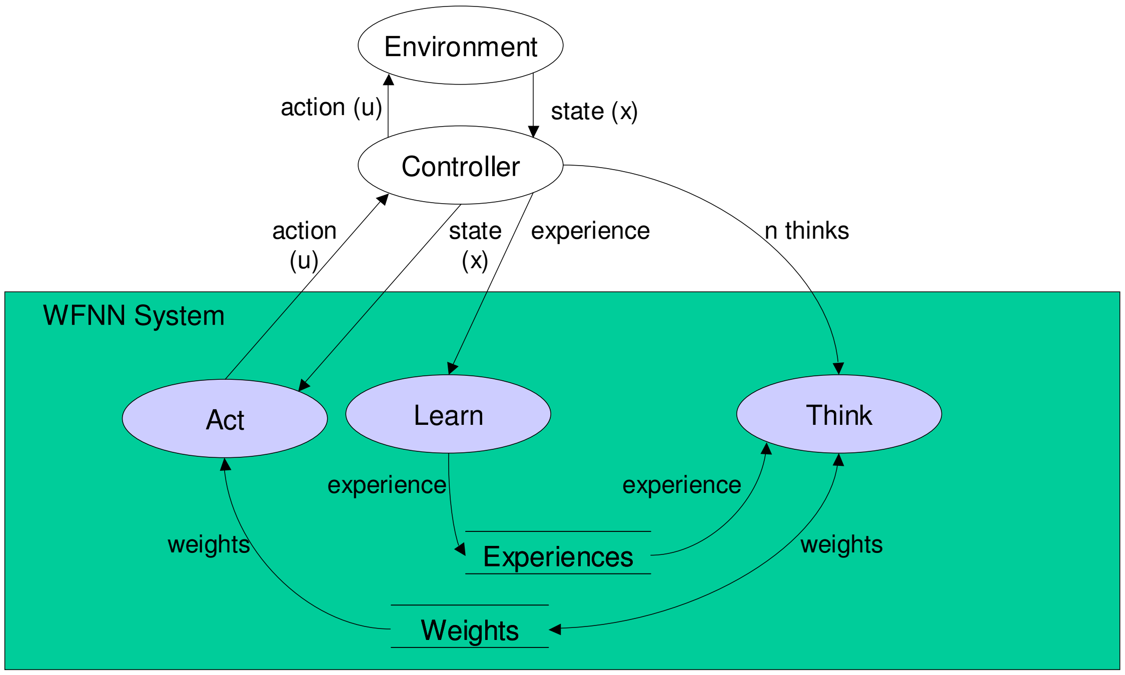


Figure 5.3: Real-time reinforcement learning timing diagram. Text in *italics* indicates desirable conditions.



```
repeat {
  measure state (at regular time interval)
  act(state, action)
  perform action
  if the state data is valid then
    learn(prev_state, prev_action, state, reward)
  think(thinks)
}
```

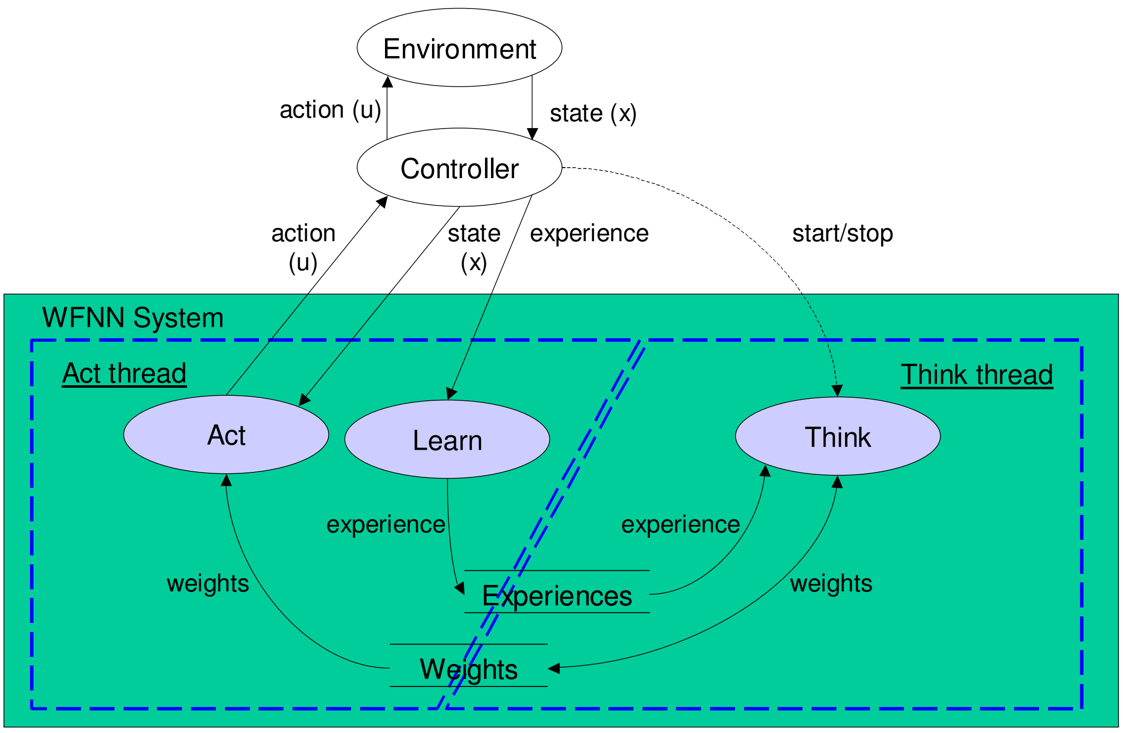
Figure 5.4: DFD and pseudocode example for the *act-learn-think* architecture

mobile robot experiments reported in Chapter 6 fixing the learning processing time is no longer appropriate. The other processing time varies and there is a hard real-time deadline: processing must be finished in time for the next video frame. The standard video frame rate is 30Hz. In the mobile robot experiments actions are executed every three video frames; thus total iteration time should be three video frames, or 0.1 seconds.

A simple adaptive approach to the problem of maintaining synchronisation is to control the number of experience replays. If the total iteration time is too long (four video frames in this example) variable `thinks` is increased; if the total iteration time is short (two video frames in this example) variable `thinks` is decreased. The adaptive scheme performs a reasonable job of maximising learning processing time but still results in occasional incorrect iteration times; it cannot react quickly to sudden increases in the other processing time.

A more sophisticated approach is used for the active head experiments in Chapter 7.

In the *act-learn-think-MT* (multithreaded) architecture the `think` method is executed in a separate processing thread with lower priority. The lower priority allows the processing loop to remain synchronised while maximising Learning Processing time: when new sensor data arrives `act` can begin immediately; until that time `think` is active. The principle is to separate real-time tasks from any-time tasks. A DFD description and pseudocode example for the *act-learn-think-MT* architecture is shown in Figure 5.5. The experience buffer and the neural network weights are shared between the two threads.



```
startThinking()
repeat {
  measure state (at regular time interval)
  act(state, action)
  perform action
  if the state data is valid then
    learn(prev_state, prev_action, state, reward)
}
```

Figure 5.5: DFD and pseudocode example for the *act-learn-think-MT* architecture

This approach results in more consistent loop times than the *act-learn-think* approach. Synchronisation is maintained as long as the other processing time does not overrun the total allowed iteration time. Use of multiple threads also uses the resources of dual processor machines efficiently. As the complexity of the learning system grows; through increasing numbers of state and action variables, hidden layer neural network neurons, and wires; more processing time is required for adequate learning to occur. If on-board processors are performing other time-intensive tasks, for example image processing and display, a single computer may no longer be adequate.

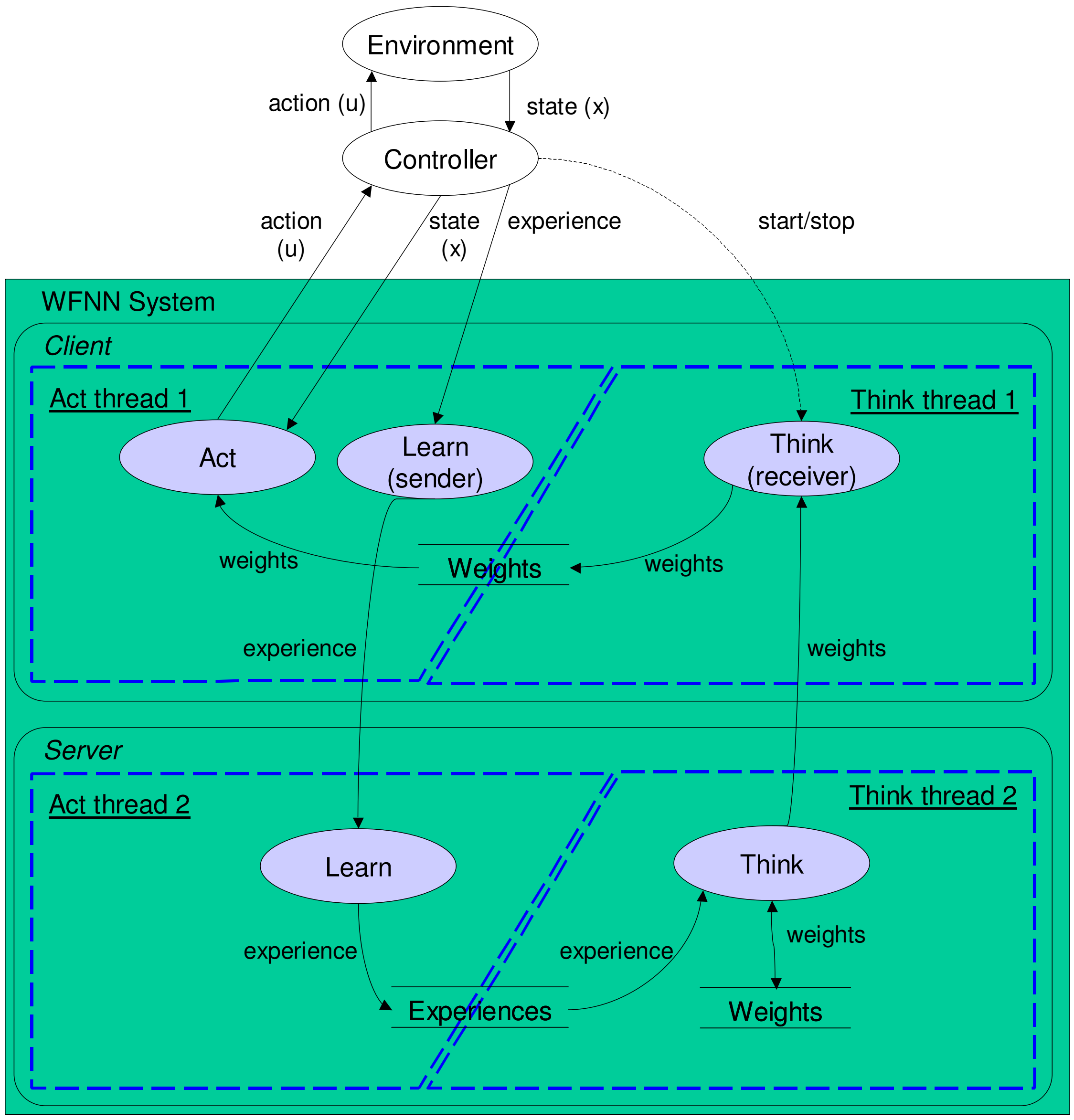
5.3 Extending Processing Power

The *act-learn-think-C/S* (client/server) architecture shares the processing load between computers connected by a network. The architecture extends the *act-learn-think-MT* architecture by moving the *think* process onto the server computer. Again the principle is to divide real-time tasks from any-time tasks; in this case the any-time tasks are processed by a separate computer. Berthouze and Kuniyoshi (1998) followed a similar approach. A DFD for the *act-learn-think-C/S* architecture is shown in Figure 5.6. The pseudocode for the client side is the same as the code for the *act-learn-think-MT* architecture (Figure 5.5).

Rather than storing experience data the client side `learn` method sends the experience over the network to the server. Rather than processing the experience data the *think* thread on the client side receives updates to the neural network weights sent from the server.

The server side is similar to the pseudocode for the *act-learn-think-MT* pseudocode. The `learn` method on the server side waits for experiences sent by the client and stores them. The *think* thread on the server processes the stored experiences and periodically sends updated neural network weights to the client. The `act` method on the server is not required for normal use but it is still functional. It can serve as an investigation interface to the learning algorithm; the current policy can be extracted by calling `act` with various states and recording the resulting actions. The similarities between the client, server and single computer architectures are exploited by inheriting both the client and server from the original single-computer class. The server side could easily be composed of more than one computer, allowing scaling to any required amount of processing power.

All network communication is through UDP/IP with CRC checking. Corruption checking via CRC checks is important since corrupted experience data could cause the learning system to diverge, and corrupted neural network weights could cause the robot to act in an unpredictable manner. If communication packets are corrupted they are discarded without retransmission. Discarding is acceptable since, unlike the *remote-brain* architecture (Inaba, 1993), the control loop in our architecture is closed on board the robot. In the *act-learn-think-C/S*, if an experience packet is lost between the client and the server the learning system has one less experience to learn from; if a neural



```
startThinking()
repeat {
    measure state (at regular time interval)
    act(state, action)
    perform action
    if the state data is valid then
        learn(prev_state, prev_action, state, reward)
}
```

Figure 5.6: DFD and pseudocode example for the *act-learn-think-C/S* architecture

network weights packet is lost between the server and the client an updated packet will be sent in less than a second. Maintaining closed-loop control is an important factor in maintaining robot safety in dynamic environments.

5.4 Safety Management

A robot's behaviour can be a danger to people, the robot itself, and other equipment. A learning robot is less predictable than a robot with hard-wired or programmed behaviour. In the experiments reported in the following chapters several techniques are applied to improve safety. Mobile robots are monitored constantly as they are operating in an environment with people and dangerous obstacles, including stairs. The experiment is terminated automatically if a contact sensor is engaged. If a small robot were used it might be possible to continue the experiment after collision, but with a larger robot collisions are dangerous and indicate irreversible system failure. Actions are filtered by setting maximum accelerations. The filters are set at a level that barely protects the hardware so motion during the early stages of learning still appears coarse. Actions that are obviously dangerous are vetoed by hand-crafted rules.

Filtering and vetoing adds hand-crafted information that can simplify the learning task. If the hand-crafting makes the task trivial, by making solution of the task inevitable, or by constraining the results so much that the solution is uninteresting, then the purpose of the learning is lost. In our experiments, we have made a distinction between the safety system and the learning system in order make it clear what has been learnt: we use sensor inputs that are not part of the state representation to detect dangerous situations, and indicate when a hard-wired vetoing behaviour is active with an audible warning. The sound warns experimenters that they are not observing a learned behaviour, and warns bystanders to be on their guard.

Although the information provided through filtering, vetoing, or otherwise not following the actions suggested by the learning system should simplify the task, it can be a detriment to the learning process. The learning system's sole source of information is experience ordered sets consisting of a state, action, next-state, and reward. When a veto occurs, the experience ordered set that occurs can be represented by,

$$\left(x_t, u_{t\,\text{actual}}, x_{t+1}, r\left(x_t, u_{t\,\text{actual}}, x_{t+1}\right)\right) \tag{5.1}$$

where the action that is actually executed (by veto) is denoted as u_{actual}; the action proposed by the learning system would be denoted as u_{proposed}; and x_{t+1} is the measured state produced after executing $u_{t\,\text{actual}}$ in state x_t. The experience is valid and useful for learning: if the learning system is capable of learning from non-estimation policy actions the action-values can be updated. However, the action-value for u_{proposed} has not been updated. Therefore, the learning system has gained no information that will dissuade it from attempting to execute u_{proposed} in the future. In simulation experiments these conditions lead to the expected value of vetoed actions rising to the maximum representable value. The distortion prevents learning.

If the robot can learn from executing marginally unsatisfactory actions it can learn to avoid executing more seriously dangerous actions (Saito and Fukuda, 1994). The appropriate experience ordered set is:

$$\left(x_t, u_{t\,\text{proposed}}, x'_{t+1}, r\left(x_t, u_{t\,\text{proposed}}, x'_{t+1}\right)\right) \tag{5.2}$$

where x'_{t+1} is the measured state produced after executing $u_{t\,\text{proposed}}$ in state x_t. This experience is not available as x'_{t+1} and $r\left(x_t, u_{t\,\text{proposed}}, x'_{t+1}\right)$ are unknown, since u_{proposed} was never executed.

Another approach to providing an experience with the vetoed action is to provide a fabricated terminating experience with a strong negative reward,

$$\left(x_t, u_{t\,\text{proposed}}, r = -1\right) \tag{5.3}$$

This approach introduces a discontinuity into the reward function which reduces stability. In simulation experiments the learning system did not successfully generalise between vetoed and similar slightly unsatisfactory actions when using this system. The consequence is that the learning system routinely executes actions that are almost unsatisfactory enough to be vetoed.

A compromise between the valid experience in Equation (5.1) and the unavailable but desirable experience in Equation (5.2) is the experience:

$$\left(x_t, u_{t\,\text{proposed}}, x_{t+1}, r\left(x_t, u_{t\,\text{proposed}}, x_{t+1}\right)\right) \tag{5.4}$$

where x_{t+1} is the measured state produced after executing $u_{t\,\text{actual}}$ in state x_t. If the vetoing system were not part of the learning environment, the experience would be invalid, but otherwise it is valid. If,

$$r\left(x_t, u_{t\,\text{proposed}}, x_{t+1}\right) < r\left(x_t, u_{t\,\text{actual}}, x'_{t+1}\right) \tag{5.5}$$

then u_{proposed}'s value will drop below the value of u_{actual} and u_{proposed} will no longer be considered the most desirable action. Even if Equation (5.5) is not always satisfied the system works well in practice. It causes no discontinuities in the reward function and prevents $\mathcal{Q}\left(x, u_{proposed}\right)$ from continuing to rise.

Processing architectures that separate *acting* from *learning* allow calling the `learn` method twice, allowing the learning system to learn from the experiences in both Equations (5.1) and (5.4),

$$\begin{gathered}\left(x_t, u_{t\,\text{actual}}, x_{t+1}, r\left(x_t, u_{t\,\text{actual}}, x_{t+1}\right)\right) \quad \text{and} \\ \left(x_t, u_{t\,\text{proposed}}, x_{t+1}, r\left(x_t, u_{t\,\text{proposed}}, x_{t+1}\right)\right)\end{gathered}$$

Intuitively, this means that both the proposed action, and the action that was actually executed, led to state x_{t+1}. If the reward for the proposed action is lower than the reward for the action actually executed then the actual action will be preferred to the proposed action.

The approach of calling the learn method twice is used in the mobile robot experiments reported in Chapter 6. If the robot's infrared sensors detects a nearby obstacle, then any actions proposed by the learning system that would move the robot towards the obstacle are vetoed. In this situation the vetoing system produces an actual action of zero-motion and sounds an audible warning. The learn method is called twice, once with the veto system's zero-motion actual action, than again with the learning system's proposed but dangerous action:

```
learn(prev_state, zero_motion_action, state, reward)
learn(prev_state, learnsys_action, state, learnsys_action_reward)
```

5.5 Artificial Neural Network Implementation

The artificial neural network used in the experiments is a feedforward network, or multilayer perceptron, with one hidden layer. It is trained through incremental backpropagation (Rumelhart et al., 1986). Tansigmoidal neurons are used throughout. Time profiling of the learning code during early simulation experiments revealed that over 90% of processor time during learning is consumed in calculating the hyperbolic tangent function. The built-in `tanh` function was replaced with a function based on a fast exponentiation method (Schraudolph, 1999), resulting in an almost tenfold increase in overall program speed.

The neural network implementation supports multithreading so that the *act* and *think* threads in architectures *act-learn-think-C/S* and *act-learn-think-MT* can safely use the neural network simultaneously. Methods for saving and loading the neural network weights allow the status of the learning system to be stored for future use or analysis. The *act-learn-think-C/S* architecture supports this ability on both the client and server. Saving and loading of weights is used in Chapter 7 to produce graphs representing the learned policy.

5.6 Wire Fitter Implementation

C++ source code for the wire fitter is included in Appendix B. This source code has been included as it is the mathematical foundation of the WFNN system. It implements Step 3b from the WFNN algorithm in Chapter 3, including evaluation of Equations (3.1), (3.2), (3.3), (3.4), and (3.5). The wire fitter is inherently thread-safe as it has no variables, only constants.

5.7 Summary

Reinforcement learning algorithms require strict timing for sensible real world operation. Synchronisation with sensor data must be maintained. Separating real-time activities from any-time activities is a useful approach for real-time design. The idea leads to multithreaded and client/server architectures that improve processor utilisation as well as improving timing. These architectures are scalable, allowing more processor power to be added for the processor intensive activity of neural network training.

The architectures proposed were developed incrementally, initially based on the results of simulation experiments, then followed by real world experiments. The results of the simulation experiments led to architectures that allow acting without learning, and passive learning. The real world experiments emphasised the importance of real-time performance and safety management.

Maintaining safety is a nontrivial aspect of applying reinforcement learning techniques to robots. Filtering and vetoing actions in the pursuit of safety interferes with the learning process and can encourage injurious actions. Successful vetoing is possible only if the learning system has the opportunity to learn from its mistakes. Based on the results of simulation and real world experiments a method was proposed that allows vetoing and learning from mistakes without introducing destabilising discontinuities into the reward function.

The architectures, algorithms, and methods proposed thus far are applied to mobile robot and active head learning tasks in Chapters 6 and 7.

Chapter 6

Mobile Robot Experiments

A monk asked, "Is a baby that is just being born endowed with the six consciousnesses or not?"

The master said, "A ball thrown into a rushing stream."

Joshu, c. 890, no. 243

Earlier chapters proposed the WFNN algorithm and evaluated various aspects of its performance using simulation tasks. As stated earlier, success in simulation does no guarantee of success in the real world (Brooks and Mataric, 1993).

This chapter applies the WFNN algorithm to learning vision-based behaviours, target pursuit and wandering, on a real mobile robot in real-time. The task includes behaviour coordination; choosing a suitable state, action and reward representation; vision processing; managing real-time issues; and safety management. The use of the WFNN algorithm eliminates the need for camera calibration and system modelling. Unlike supervised learning approaches there is no need for a teacher to present ideal behaviour. The results show that the WFNN algorithm is suitable for robot control tasks and demonstrate the difficulty of learning in real environments.

6.1 Vision-Based Robot Behaviour

A robot's task is to intelligently interact with the physical world. Sensing is a necessity in all but the most constrained environments. Contact-detection, infrared, laser, and sonar sensors are insufficient to interact with complex environments and people with the grace of an animal—vision is required.

In the animal kingdom vision reduces the need for physical contact, allowing accurate, passive sensing at a distance. Using vision, animals can navigate, find food, avoid predators, and find a mate. Vision reveals the structure of the surrounding world, including the position and motion of objects. Animal vision hardware ranges from cells that detect light intensity without direction, through to compound, pinhole, lens, double, and even mirror-equipped eyes (Dawkins, 1996).

6.1.1 3D Reconstruction and Planning

Most early work in robotics was characterised by the *sense-model-plan-act* cycle. The approach was to sense the state of the environment, model the state of the environment, plan a sequence of actions to achieve some task, then act out the plan. This is an application of the classical AI planning approach pioneered by Newell and Simon (1963).

The predominant early approach to vision was to sense the world through a single still picture, then attempt to construct a full 3D model based on the image. It culminated in the computational vision theory of Marr (1982) who defines a hierarchy of representations from the image through to object descriptions. This could be called the *general* approach to vision; if it were obtainable, full representation should be sufficient for *any* vision-based task. A robot with a general vision and planning system can cope with unexpected events by simply sensing the situation (e.g. detected child); updating its world model (e.g. add child to current room); generating a plan (e.g. put green block on red block, put child on green block); then executing the plan. However, implementation of this type of system has been more difficult than expected.

Most early approaches to machine vision were hierarchical, either top-down or bottom-up (see for example Roberts, 1965; Barrow and Tenenbaum, 1981; Marr, 1982; Ballard and Brown, 1982). A top-down approach searches for objects that are expected to be in the image based on a world model. Unexpected or unmodelled objects can go undetected. Bottom-up approaches start with the image and work upward towards a model. Without any model of what to expect, the bottom-up process has trouble separating trivia from information.

The full 3D reconstruction approach relies heavily on camera calibration (Tsai, 1987). Camera calibration consists of finding parameters for a model that will form a mapping between the image plane and world coordinates. Inaccuracies in the calibration and errors in interpreting the scene will cause errors in the construction of the 3D model. After lower level modelling, the sense-model-plan-act approach usually converts the 3D reconstructed model into a symbolic, high-level representation so that a plan can be created based on the model, assuming that the model is complete and correct. The plan is then executed. After the plan, or some steps of the plan, have been executed, the sense-model-plan-act cycle begins again.

The fundamental flaw in these approaches is to presume that the world is easy to model and easy to interact with (Moravec, 1990; Brooks, 1991). The planning approach is capable of producing complex solutions to complex problems—but if a robot cannot cope with someone walking through the scene the plans will never be executed successfully.

To cope with the deficiencies in these approaches (and the lack of processing power at the time) researchers often constrained the world through controlled lighting and simplifying the world to walls and static geometric objects. Algorithms that operate in these simplified environments will not necessarily scale to more complex, dynamic environments. SHAKEY was a pioneering example of the sense-model-plan-act in a simplified world approach (see Nilsson, 1984). SHAKEY was the first mobile robot to navigate using vision, it was designed at the Stanford Research Institute in the late 1960s. SHAKEY's goal was to push large geometric objects. Early versions of SHAKEY tried to execute the entire plan without monitoring for success or failure; as a consequence failure was almost inevitable.

The Stanford Cart operated in less constrained environments: obstacles included chairs and potted plants and attempts were made to operate outdoors (Moravec, 1990). The Stanford Cart was developed at Stanford Artificial Intelligence Laboratory in the mid 1970s. The system was the first to use stereo vision to successfully navigate a robot in an environment with obstacles. Instead of using two cameras the Cart used a single camera that could be moved along a rail into nine positions. The image processing system used features to identify objects. Another significant idea was to model obstacles generically, by using spheres to represent safety boundaries, rather than with full geometric models. This led to the ground-breaking ability to avoid real obstacles, such as chairs; although shadows were often detected as obstacles too. Ironically, the robot had trouble detecting traditional artificial obstacles: white cubes and trees made from cardboard.

The Cart's image processing and planning steps were time-consuming, taking around 15 minutes. This obviously precluded avoiding dynamic objects. After planning the Cart would execute 75cm of the plan, then stop, obtain new images, and re-plan. The approach acknowledged that errors are bound to occur. Moravec (1990) concluded that skills humans can perform instinctively are much harder to imitate than intellectual skills.

6.1.2 Purposive Vision and Behaviour

The indisputable lesson from the early vision-based robotics experiments is that attempting to model the world, then executing a complex plan open-loop (without sensing), cannot succeed in a complex dynamic environment. Irrespective of the robot architecture, at some level the robot must act in a closed-loop fashion using sensor data. Closed-loop robot control requires fast, robust sensing, and a fast method of choosing actions.

It is now commonplace to perform operations on video frames in real-time. Processor power has increased dramatically; CCD cameras have become cheap thanks to the ubiquity of video cameras. Apart from changes in hardware, there has also been a change of approach. The 3D reconstruction and planning paradigm demanded too much information from single images and expected too much intelligence from a computer. The computer vision problem is more difficult than expected—the ease with which people perform tasks with vision is misleading.

Instead of trying to fully model the world, purposive approaches seek to extract only the information that is required for a specific behaviour (Ballard and Brown, 1992). Instead of trying to extract as much information as possible from a single image, newer methods use temporal information. Purposive systems often have non-hierarchical architectures and combine top-down and bottom-up processing. The new approach has less spectacular aims compared with older approaches and has a higher regard for animal vision systems and the difficulty of the problem.

Marr stated that animals could not all be using the same general vision representation; they are likely to use one or more representations that are tailored to the animal's purposes (Marr, 1982, p. 32). However, Marr's work was focused on human abilities, and he argued for general representations in computer vision in the hope of replicating these abilities. Robots still struggle to perform tasks that insects can perform, such as navigating and avoiding obstacles (Srinivasan and Venkatesh, 1997; Franz and Mallot, 2000). It seems impractical to attempt to replicate human abilities on a robot through general representations before the insect level of competence has been achieved. Animal's vision systems are tailored to their lifestyles: herring-gull chicks assume that anything with a red dot is their parents bill; while sticklebacks assume that anything red is an enemy stickleback, even if it is the size of a van (Tansley, 1965). Shortcuts and simplifications are adequate behavioural cues for animals that find food, navigate, avoid enemies, find mates, and bring up their young—they should be enough for a robot with insect-level competence.

Horswill's (1994) Polly is an example of a purposive, vision-based system. Polly was a mobile robot that used simple visual cues, intensity boundaries, to detect obstacles. Rather than making strong assumptions about the type of obstacles present, the system made assumptions about the appearance of safe areas, as appropriate for its environment. Polly was highly responsive because of the tight closed-loop connection between low-level vision and behaviour.

Most purposive systems still rely on the ability of their designers to foresee the situations that will be encountered. Not only is the repertoire of behaviours fixed, the behaviours themselves are fixed. Our ability to change our behaviour based on various forms of feedback is missing from this type of robot. Instead, the robot designer relies on designing behaviours robustly, and adequately predicting the range of problems the robot will encounter.

Reinforcement learning is one path to introducing novel behaviour and adaptability to purposive systems. It can reduce the reliance on calibration, and result in novel solutions to robotics problems. Like the classical planning approach, reinforcement learning systems can produce multiple step plans. Unlike the classical planning approach, reinforcement learning does not require a full, symbolic representation of the environment, nor a model or simulation of the system. Rather, actions are re-planned at every time step in closed-loop, real-time response to perceived changes in the environment, based on previous experience operating in the environment.

The remainder of this chapter describes experiments in applying the WFNN algorithm to a mobile robot.

6.2 Robot System Architecture

The research platform was a Nomad 200 with a Sony EVI-D30 colour camera. The camera pointed forward and downward (see Figures 6.1 and 6.2). The Nomad 200 is capable of forward-backward translation and rotation.

The system architecture was based on the behaviour-based system reported by Cheng and Zelinsky (1996). Their system built on simple, low-level behaviours, and scaled to more purposeful behaviours such as finding a goal in a cluttered environment and cooperative cleaning (Cheng and Zelinsky, 1998; Jung et al., 1997).

We simplified the navigation system to two behaviours: *wandering* and *target pursuit*. The behaviours controlled the translational and rotational velocity of the robot. From an observers point of view, the behaviour of the robot was to forage for interesting objects.

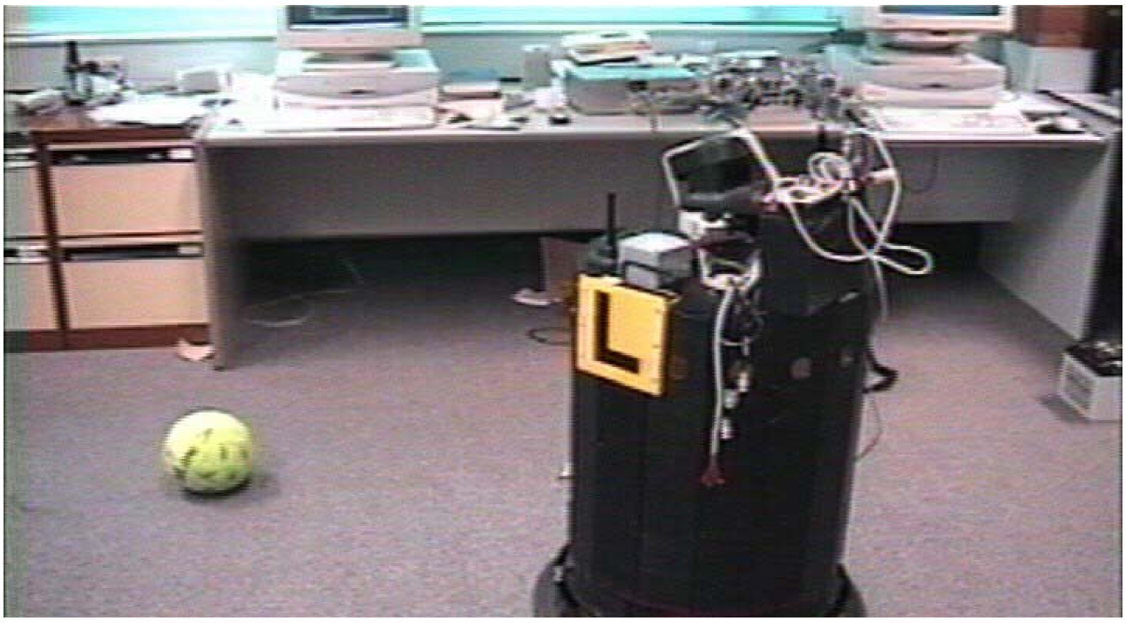


Figure 6.1: The Nomad 200 learning target pursuit. The ball is the current target.

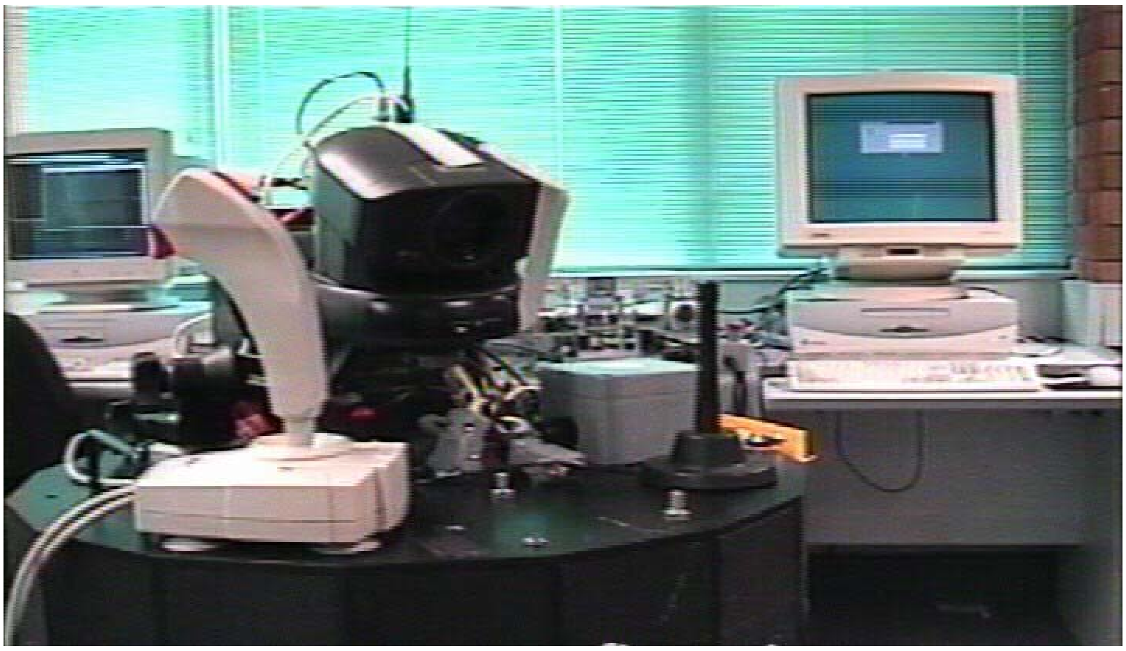


Figure 6.2: Uncalibrated camera mounted on the Nomad 200

The foraging behaviour was the result of combining the wandering and target pursuit behaviours. The wandering behaviour kept the robot moving without colliding with obstacles. If, during wandering, an interesting object was identified the target pursuit behaviour became dominant. The target pursuit behaviour moved the robot towards the interesting object. If the interesting object was lost from view then the robot resumed wandering. Figure 6.3 shows an augmented finite state machine representation of our behaviour-based system (see Brooks, 1986).

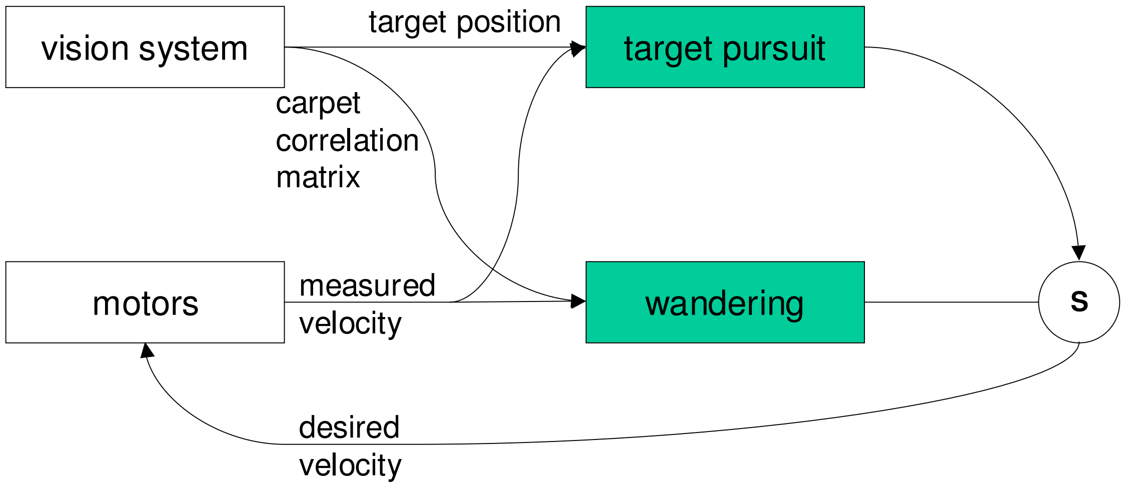


Figure 6.3: Behaviour-based model. Target pursuit dominates wandering when a target has been identified. Both behaviours use visual and motor velocity data.

Successful wandering requires the ability to avoid obstacles. The direct approach is to identify obstacles. For example, Moravec (1990) generalised obstacle avoidance by detecting and modelling obstacles generically, visually detecting corners to identify obstacles then modelling the obstacles using spheres. Cheng and Zelinsky (1996) and ourselves followed the inverse approach devised by Horswill (1994): rather than identifying obstacles, we try to identify obstacle free, navigable space. Instead of defining the expected properties of obstacles, the expected properties of free space are defined. Horswill's Polly identified free space based on the absence of edges in the camera's view. Cheng and Zelinsky's identified free space based on similarity to a stored, template image of carpet.

The free space detection approach assumes that obstacles fulfil the *ground-plane constraint* (Horswill, 1994): obstacles must rest on the ground plane, and their projection above the ground plane must be contained within the area of their contact with the ground plane. For example, a potted plant that fulfils the constraint would have a pot that does not flare outwards above the base, and the branches of the plant would not reach further outwards than the edges of the pot. The constraint is required so that height in the image plane is a strictly increasing function of distance from the robot. In practice, a safety margin is included in the navigation algorithm so that minor violations of the ground-plane constraint are acceptable.

The camera view in Figure 6.4 demonstrates our free space detection system. Smaller squares are shown for regions that are likely to be carpet, and are therefore regarded as free space.

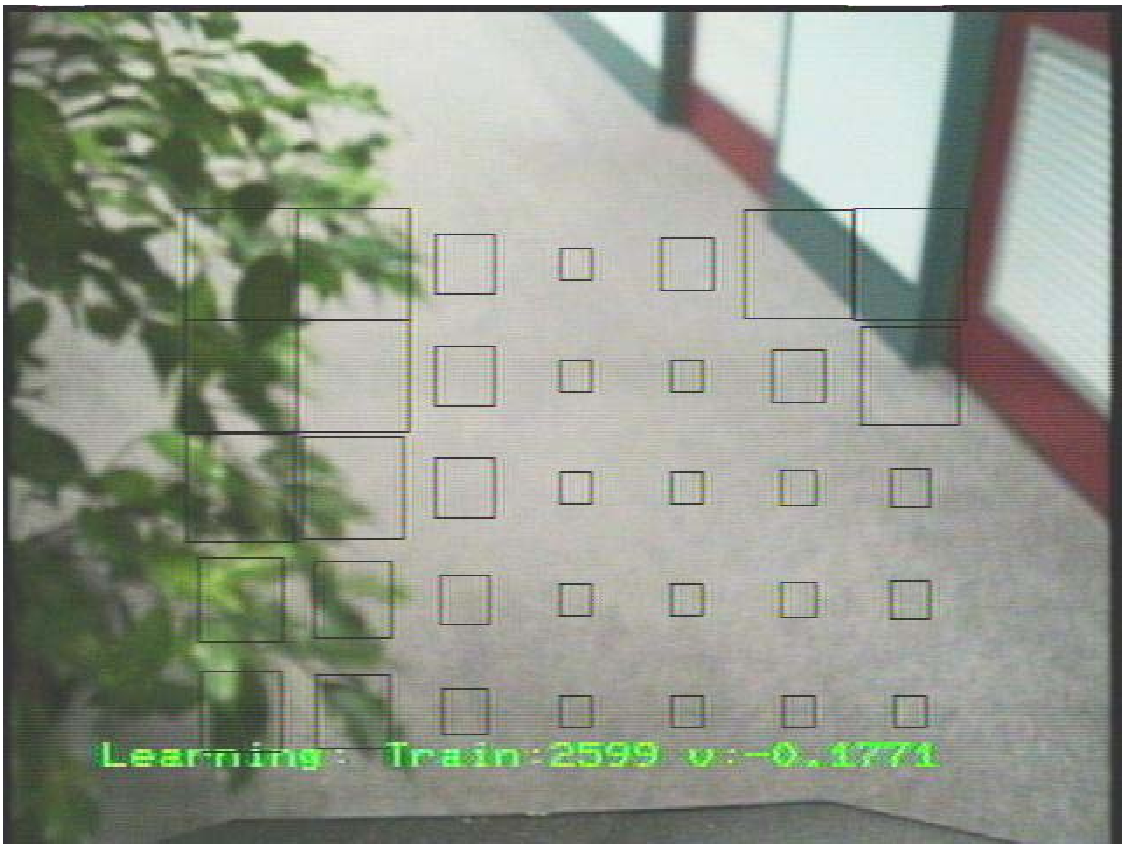


Figure 6.4: The robot's view during the wandering behaviour. Small squares indicate a good match with the carpet template; accordingly, large squares represent detected obstacles.

Targets for pursuit were identified by an *interest operator* designed to lead the robot to select isolated objects: an object was identified as interesting if it was not carpet but was surrounded by carpet. An example is shown in Figure 6.5. Once an object has been identified as interesting the robot pursues that object, whether or not it is surrounded by carpet. The goal of the pursuit was to position the object at the point in the camera's view marked by a green box in Figure 6.5. Pursuit ceased only if the target object passed out of view.

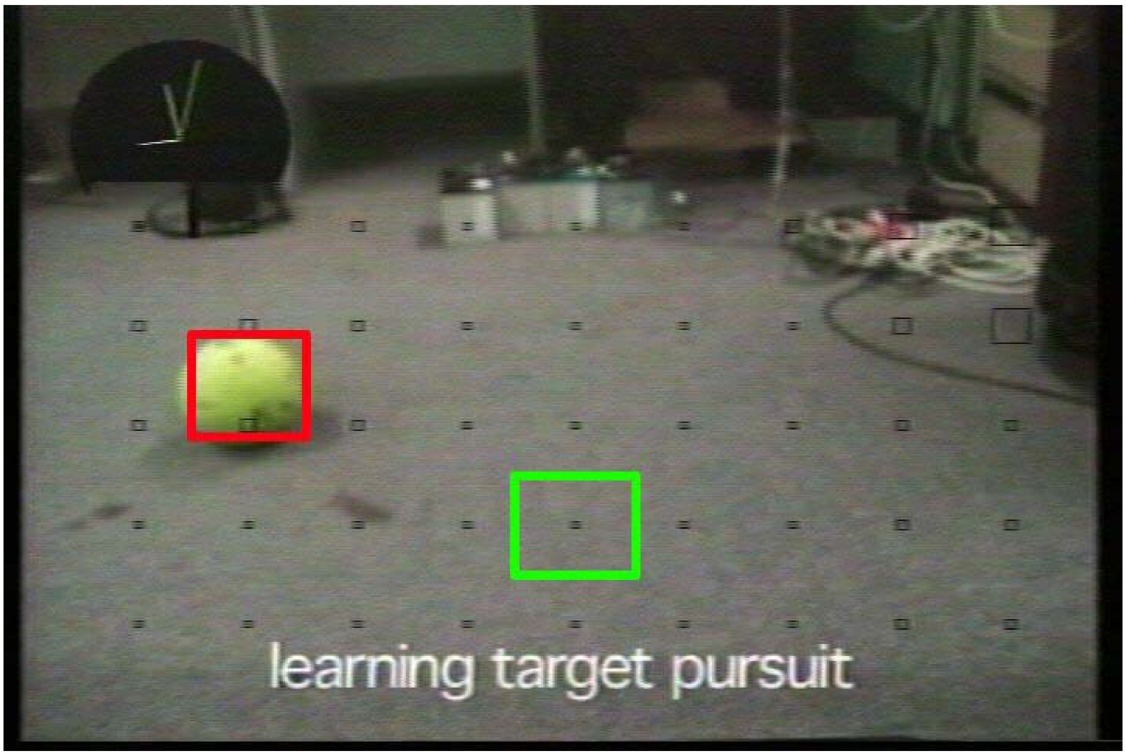


Figure 6.5: A ball, marked with a red box, has been identified as an interesting object. The green box marks the goal position for the object.

6.3 Vision System

The robot's vision system is built around *template matching* (Ballard and Brown, 1982). A small, template image is compared to regions of the robot's view using *image correlation*. Image correlation measures the degree of similarity between two images, using for example, the Sum of Absolute Differences (SAD) between the pixels over the image:

$$\text{SAD}\,(image1, image2) = \sum_{c}^{3}\sum_{x}^{m}\sum_{y}^{n} |image1(c,x,y) - image2(c,x,y)| \qquad (6.1)$$

where there are 3 colour planes, red, green, and blue; and the images are $m \times n$ pixels. A larger SAD value indicates a larger difference between the images.

For our experiments, the matching calculations were performed using a Fujitsu colour tracking vision card on-board the robot. The Fujitsu card is capable of performing two hundred 8×8 pixel SAD correlations per image frame at a frame rate of 30Hz.

The robot performed free space detection and detection of interesting objects by comparing the robot's view to stored template images of carpet. A grid of 5×7 correlations were performed. The result was a matrix indicating the likelihood that regions ahead of the robot were carpet. The likelihood is represented by the size of the squares in Figure 6.4.

Figure 6.6 shows acquisition of carpet image templates during initialisation. Different templates were acquired for use at different distances. Cheng and Zelinsky (1996) used this approach with grey-scale template matching and were able to match based on carpet texture due to the camera being close to the ground. Our system used a taller robot. Consequently, the texture of the carpet was not clear from the camera view and matching using carpet texture was not feasible. Instead, our carpet detection method was colour matching, using the colour template matching system.

The matching process coped with some change in lighting conditions; however, it was not robust enough to deal with heavy shadows or objects that were similar in colour to the carpet.

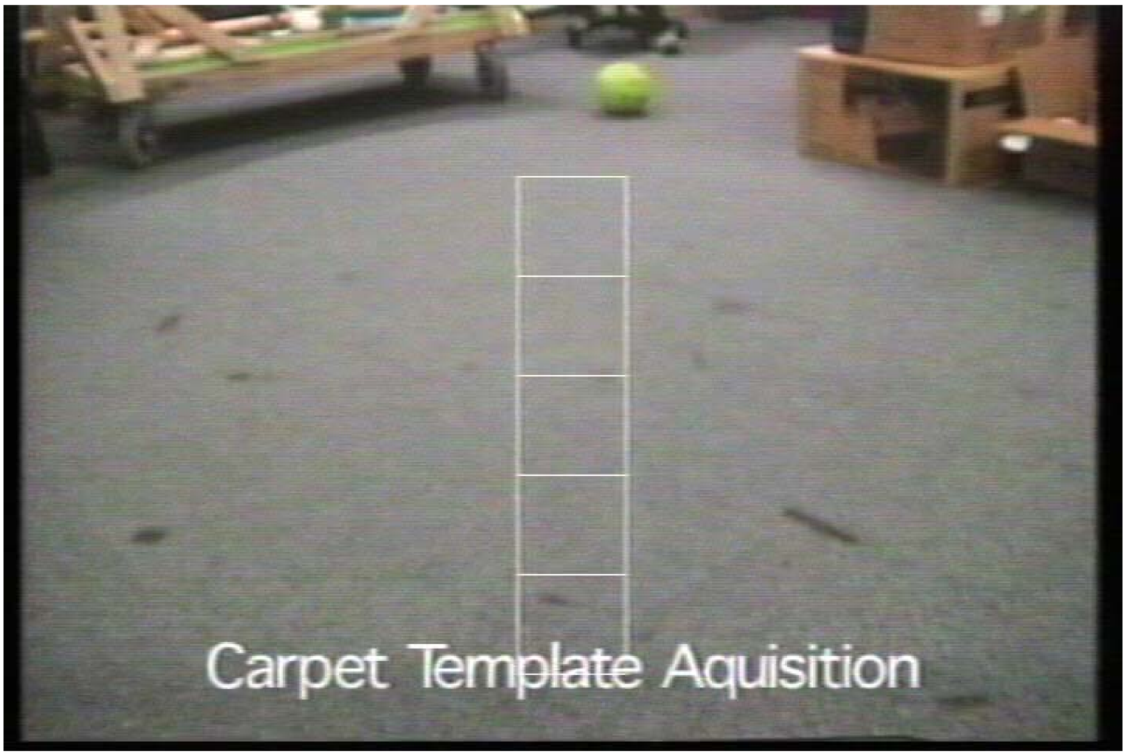


Figure 6.6: Carpet template acquisition during initialisation

Template matching is also used by the target pursuit behaviour. When an interesting object has been identified by the interest operator, a template image is taken of the object. The object is then *tracked* in future video frames by finding the correlation between the template image of the object and regions around the object's previous location. If the difference between the template and the closest match in the current image is too great then the target is presumed lost. To improve the robustness of the matching we smoothed the image by defocussing the camera slightly.

In order to control the robot a transformation is required between information from the template matching, which is in terms of pixels in a 2D image captured by the camera, into translational and rotational velocities to control the robot's motors. Cheng and Zelinsky (1996) employed a trigonometric model to perform the transformation. The model was used in both the target pursuit and wandering behaviours. Camera calibration was required, and attitude and position relative to the floor had to be measured. In our system both the wandering and target pursuit behaviours were learnt, and the camera calibration process was not required.

6.4 Learning to Wander

The purpose of the wandering behaviour was to explore an environment without colliding with obstacles. Cheng and Zelinsky's (1996) hard-wired algorithm used the matrix of correlations with the image template of the carpet to guide the movement of the robot: the robot moved in the direction of the longest uninterrupted vertical strip of carpet, or rotated if no carpet was detected (see Figure 6.7).

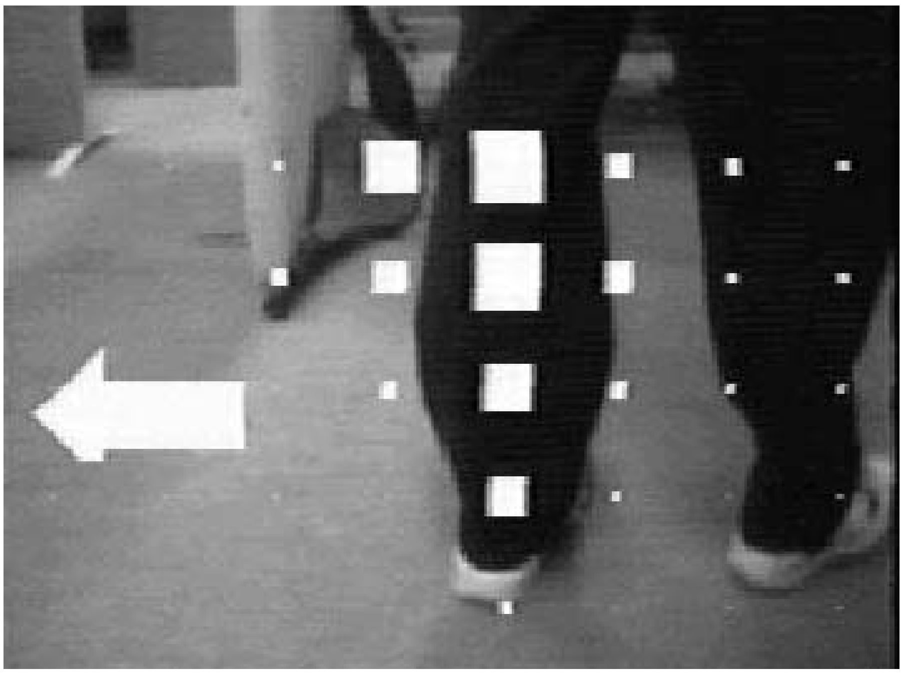


Figure 6.7: Camera view illustrating Cheng and Zelinsky's wandering algorithm. The longest uninterrupted vertical strip of carpet is on the left-hand side. Consequently, the controller moves the robot to the left.
Source of illustration: Cheng and Zelinsky (1998).

Development of our learning-based system required a formulation of the wandering task within the reinforcement learning framework. In particular, a reward function is required. There is no expert available to objectively measure the quality of the wandering behaviour and give rewards. Rather, the reward must be calculated from available data: the matrix of matches with the carpet template, and the velocity of the robot.

6.4.1 Task Representation

We conjectured that if the presence of carpet indicates free space then maximising the amount of carpet in view should keep the robot away from obstacles. Thus, negative reward should be given relative to the degree of mismatch with the carpet templates, represented in the carpet match matrix. Following this method, the optimal view for the robot is to see only carpet.

Some elaboration on the idea was required to obtain useful behaviour. If the robot is moving towards a small obstacle the reward ought to become progressively more negative. However, if each location in the carpet match matrix is weighted equally then the reward will stay constant: when the robot moves towards a small obstacle carpet becomes visible beyond the obstacle, compensating for the carpet covered by the obstacle. When the robot moves sufficiently close to the obstacle it is lost from view, hidden in the blind-spot in front of the robot. To prevent this from occurring, carpet nearer to the robot was weighted more heavily in the reward function.

With a reward function based entirely on the carpet match matrix the robot should learn to find large areas of carpet. Once it has found a large area of carpet the robot is likely to halt. Halting does not fulfil the requirements for wandering; rather, the robot is supposed to explore. To discourage halting, a positive reward was given for moving forwards. Rewarding for forward movement also punished backward motions. It is important to discourage backward motions because the robot's camera does not view the area behind the robot. Blind reversing could be avoided by banning backward motions entirely, the approach followed by Cheng and Zelinsky (1996). We gave the robot more flexibility by allowing reversing, but discouraging it through the reward function.

Figure 6.8 shows the state, actions and reward for the reinforcement learning algorithm. The reward was a weighted sum of the components shown. The final component of the reward signal punished for non-smooth motion that could be unpredictable to bystanders and possibly damage the robot. As in the simulated boat navigation experiment in Chapter 3, the reward was scaled then compressed by a hyperbolic tangent function to avoid the need to represent action-values with magnitudes higher than 1.

State	C: 5×7 carpet match matrix
	t, r: translational and rotational velocity of robot (measured by encoders)
Action	T, R: desired translational and rotational velocity of robot
Reward	$-\sum C \cdot row$: maximise visible carpet
	$+t$: move forward
	$-\{(T-t)^2 + (R-r)^2\}$: smooth motion

Figure 6.8: Reinforcement learning specification for wandering

It was difficult to obtain accurate data from the carpet matching process. Lighting varied throughout the corridors in the test area. The walls were similar in colour to the carpet. To increase reliability, if a non-carpet region was detected the region above was also considered to be non-carpet (see Figure 6.4). Additionally, the carpet template matches were smoothed with their neighbours horizontally and smoothed between frames. The filter can be represented as follows:

$$\begin{aligned}\text{smoothed}_{row,col} = (1-\alpha)\max\Big[&(1-2\beta)\,\text{raw}_{row,col} + \beta\text{raw}_{row,col-1} + \beta\text{raw}_{row,col+1}\;,\\ &\text{smoothed}_{row+1,col}\Big]\\ &+ \alpha\text{smoothed}\,(t-1)_{row,col} \qquad (6.2)\end{aligned}$$

where α and β control the degree of filtering. The formula was modified where appropriate for the extreme left and right columns and the lowermost row. These processing steps raised the quality of the state data to a standard where learning was possible.

The simple uniform-random exploration strategy described in Chapter 3 is unsuitable for use on real robots. It generates unacceptably harsh actions. Instead we added noise with a Gaussian distribution to the action suggested by the learning system. The addition generated some exploration without causing the robot to unduly shake due to coarse motion commands.

It was necessary to provide a safety buffer during learning to prevent damage to the robot. The infra-red ring on the Nomad was used to veto actions that would immediately cause collisions. The method was as described in Chapter 5. When a veto occurred there was an audible warning so that observers were aware of intervention by the hard-wired behaviour. To ensure that the vision data was being used by the learning algorithm the infra-red readings were not part of the state representation.

6.4.2 Results

At the beginning of the trial the robot's behaviour was random; initialisation of the neural network with small weights produced slow movements. Unless the learning process began while the robot is directly facing a wall, it quickly learnt that forward velocity brings rewards. When the robot reached a wall it found that moving forward is not always rewarding and adapted its behaviour accordingly. As learning continued the number of action vetoes reduced, but they still occurred when the robot encountered an obstacle that was not detected through vision.

Figure 6.9 shows a path recorded during a wandering experiment. Actions were executed at 10Hz. Learning began at the point marked *start*. Movement became smoother and more consistent throughout the trial. The quality of the learned wandering algorithm was variable. Often the robot had a preference for turning in one direction. When faced with an obstacle the robot might always turn left, even if the obstacle was on the left-hand side. If the robot gets close to the obstacle it might continue to turn left but reverse slowly. The behaviour was rather uninspiring; however, it did achieve the set task. It was likely that the robot acquired the behaviour if early in training the robot encountered more obstacles on one side than the other. Pomerleau (1993) observed a similar problem training a supervised learning algorithm to drive a vehicle: if the algorithm was trained with more left turns than right it developed a bias.

6.5 Learned Visual Servoing

The target pursuit behaviour is composed of an interest operator, that selects a target object, and the process of *visual servoing*. Visual servoing is closed-loop control of some part of a robot based on visual information (Hutchinson et al., 1996). The closed-loop approach leads to greater accuracy than the older *look-then-move* approach as corrections are made continuously. Visual servoing requires the ability to track the position of some object using vision, and to control some effector based on feedback from the tracking.

There are two basic approaches to the control component of visual servoing: position-based and image-based. Both generally require calibration. In position-based systems an error signal is defined in the robot's coordinate system. A model describing the relationship between image coordinates and the robot's coordinate system is required. Sometimes it is possible to learn the model (Hashimoto et al., 1991; Park and Oh, 1992; Lo, 1996; Buessler et al., 1996). Such learning systems are suitable for servoing manipulator arms, where joint angles define the position of an effector.

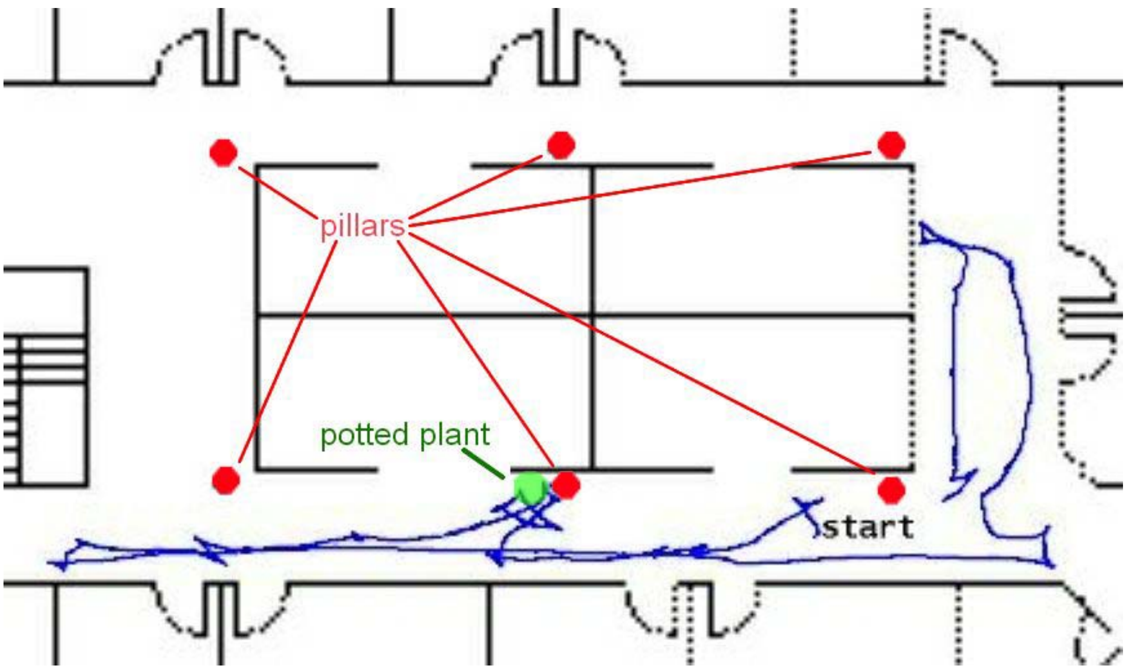


Figure 6.9: A path from a wandering experiment in an open plan office. The robot began at the point marked *start*, learning from scratch. Several pillars and a potted plant were avoided. Total trial time was 25 minutes, or 15637 actions. The trial ended when the robot clipped a pillar that was not within view and not detected by the infra-red 'bumper'. The target pursuit behaviour was disabled for this experiment. The movement trace is from uncorrected odometry data. As a consequence there is some error in the trace.

In image-based systems the error signal is defined in image space. The inverse of the image Jacobian is used to relate desired incremental change to the image to changes in actuator settings. It is possible to learn an approximation of the Jacobian (Hosoda and Asada, 1994) but this is complicated since it can vary with the state, for example the distance of the target object from the camera (Wu and Stanley, 1997; Hutchinson et al., 1996). It is possible to simplify calculation of the Jacobian and other calibration processes by exploiting the physical characteristics of the system, for example being able to rotate the camera to see how the target moves (Yoshimi and Allen, 1994).

Our approach was to learn a direct mapping from image space and other state information to actuator command using continuous state and action reinforcement learning. As the robot looked at a 3D world through a monocular camera it was assumed that all targets were close to floor level. The reinforcement learning method may not be able to achieve the performance of well calibrated systems but gives greater flexibility—if the camera is bumped or moved or the actuator configuration is changed the system continues to function.

6.5.1 Task Representation

The task of the visual servoing system is to position a target object at the goal position. The reward signal reflects the task. Negative rewards are given relative to the error in the position of the target object. Figure 6.10 shows the state, actions and reward for the reinforcement learning algorithm for learned visual servoing. The reward function included a term that punishes for energy consumption. This helped to reduce the robot's reaction to minor changes in the target object's position due to tracking noise. There was no punishment term for target velocity due to excessive noise in the target velocity estimation process.

State	x, y: pixels error to target
	Δx, Δy: pixels/second velocity of target
	t, r: translational and rotational velocity of robot (measured by encoders)
Action	T, R: desired translational and rotational velocity of robot
Reward	$-\sqrt{x^2+y^2}$: movement to target
	$-T^2 - R^2$: saving energy
	$-\{(T-t)^2 + (R-r)^2\}$: smooth motion

Figure 6.10: Reinforcement learning specification for visual servoing

6.5.2 Results

Actions were executed at 10Hz. The robot learnt to servo in real-time to a target reliably after about 15 minutes of learning. This includes time spent wandering (looking for targets). During that time around 1500 state, action, next-state, reward experiences were gathered. Figure 6.11 shows an image sequence from the robot's view while learning to servo. The robot had not learnt an adequate policy yet and overshot the target. The position of the selected object is shown by a red box. The goal position is shown by the green box in the lower middle of each image. The sequence is also included as a movie file on the CD-ROM that accompanies this thesis. When the camera was mounted facing straight ahead the learned behaviour was to turn toward the target while driving forward until the target was at the goal position. The robot slowed down as it got closer to the target. The robot also learnt to reverse if the target was too close to the robot. Figure 6.12 shows trajectories of various targets in image space. Examples of step response and motor commands are shown in Figure 6.13. The robot did not move fast

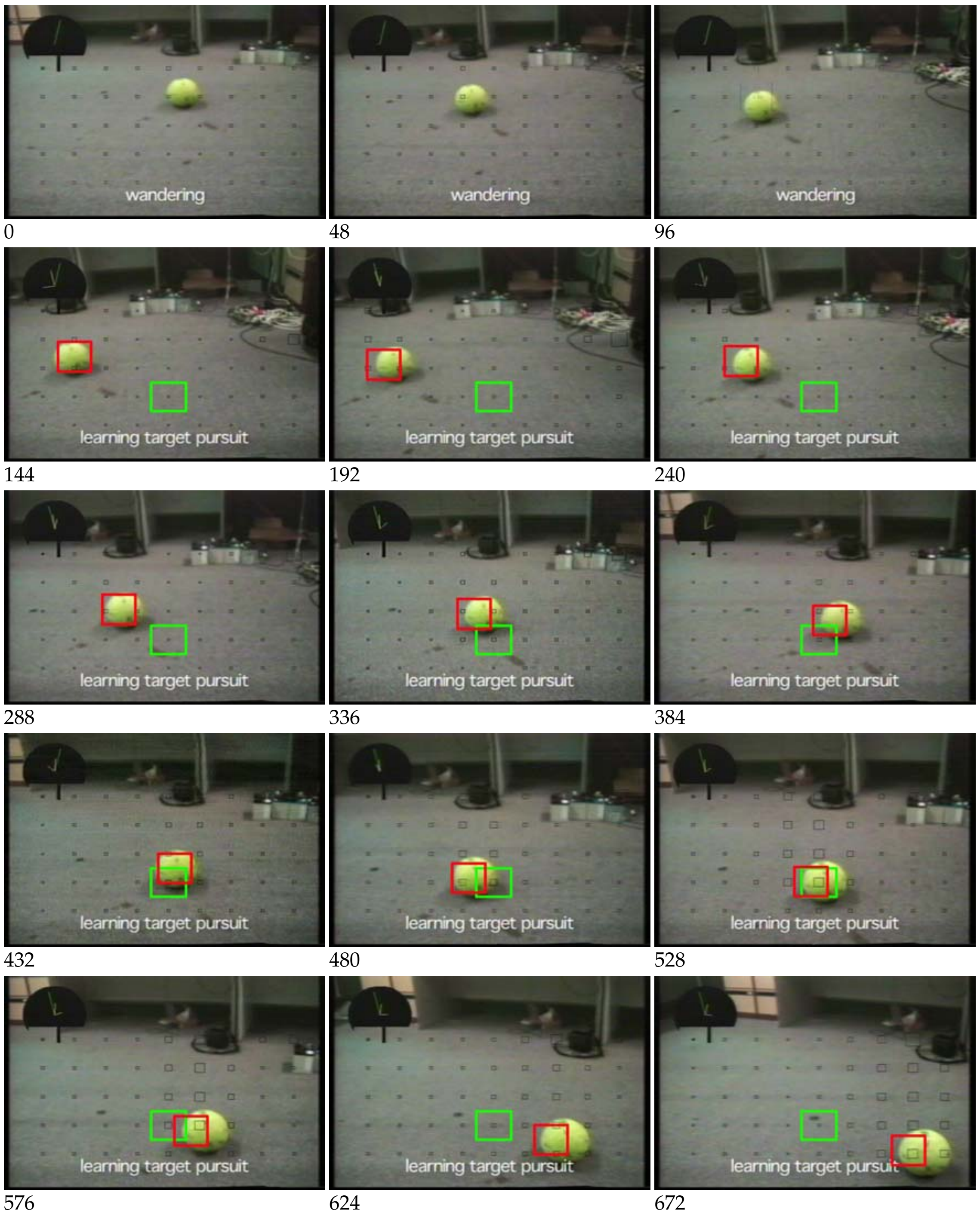


Figure 6.11: Learning to visually servo. Images are from the robot's view. The robot was wandering until the interest operator chooses the ball as a target (*frame 144*). The robot had not learnt a good strategy yet so it lost the target. The numbers below each image indicate the number of frames at 30Hz; therefore images are around 1.6 seconds apart.

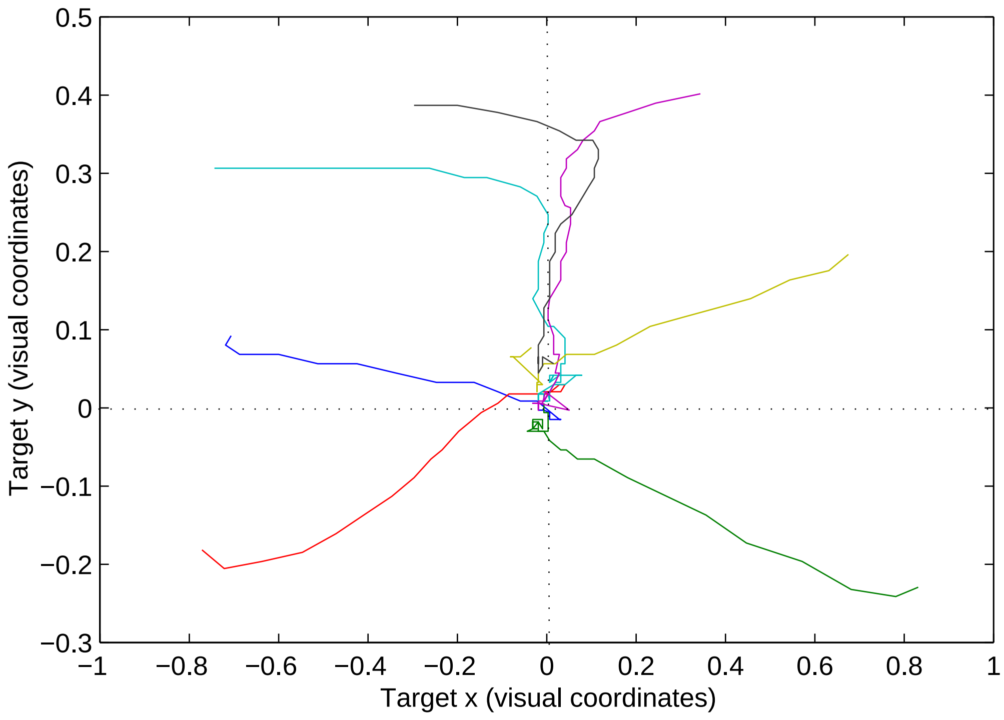


Figure 6.12: Example target trajectories

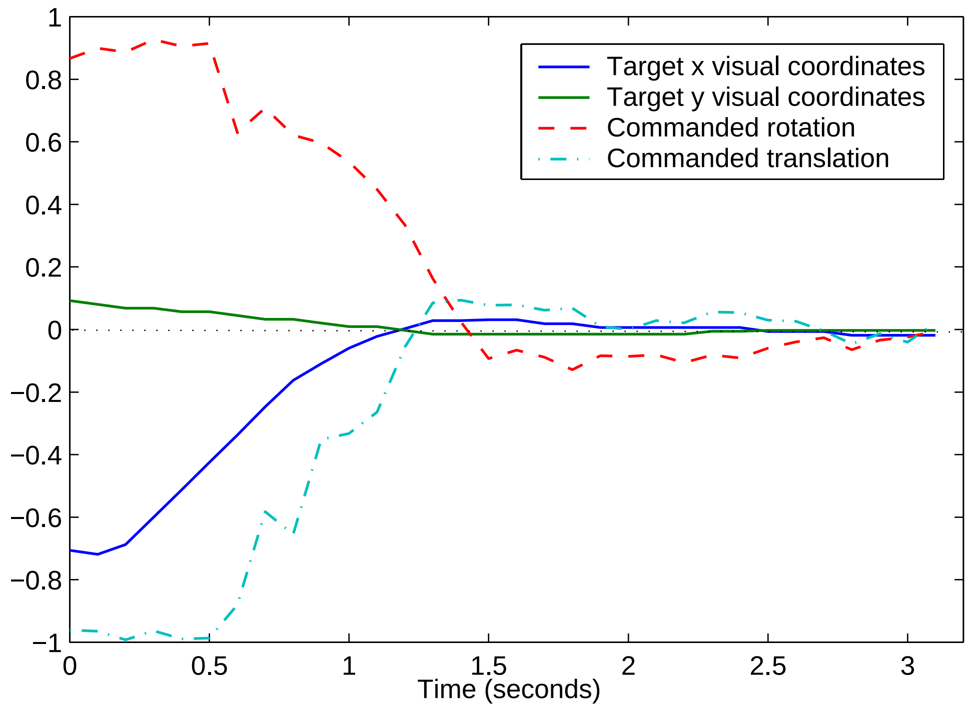


Figure 6.13: Example Step response and motor commands

enough to assess whether the learning system was compensating for the speed of the target.

The robot was also able to learn to servo successfully even if the camera was misaligned. With a misaligned camera the robot was often unable to turn the wheels in the direction of the target without moving the target into a worse location or losing the target completely. The learning system instead developed a zig-zagging method to move toward the target. The camera mounting is shown in Figure 6.14. Figures 6.15 and 6.16 show pursuit of a moving target with the camera misaligned. The robot's view is inset. The sequence is also included in a movie file on the CD-ROM.

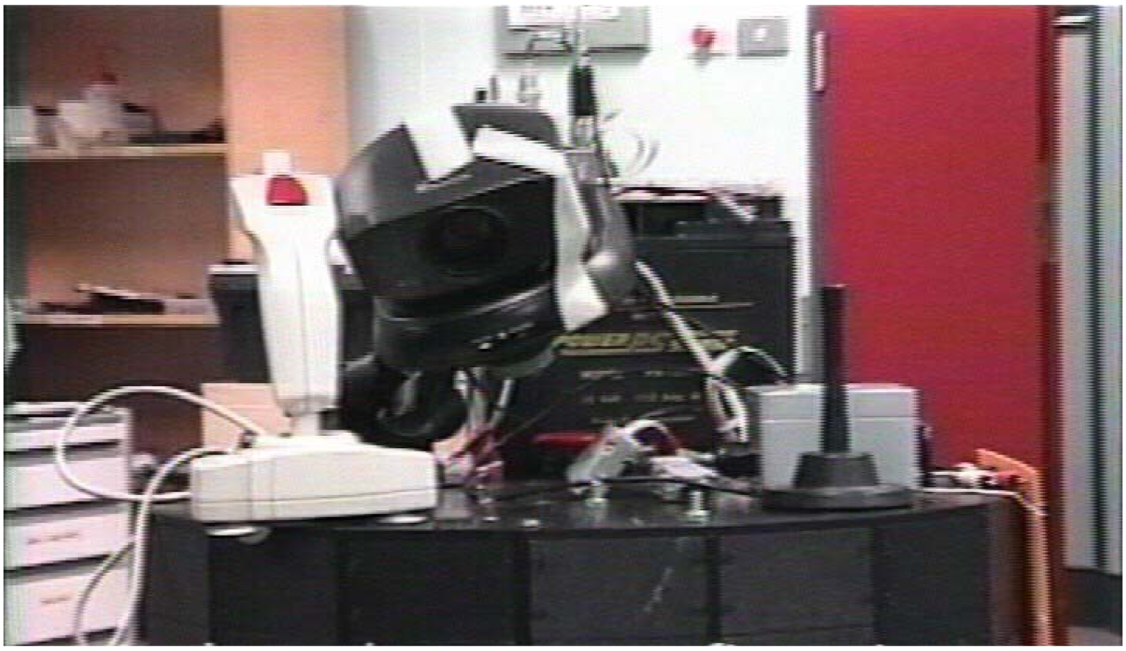


Figure 6.14: The camera is misaligned, pointing in a different direction from the translation axis

6.6 Application of Off-Policy Learning

Off-policy learning techniques were described in Chapter 2. Two off-policy learning techniques were used in the mobile robot experiments: *experience replay*; and *learning from other behaviours*.

Learning from already working behaviours reduced the period of random behaviour that would require close supervision. In this experiment data gathered while servoing was provided to the learning to wander system. The servoing system learnt from the actions of the wandering behaviour in the same way. Implementing learning wandering

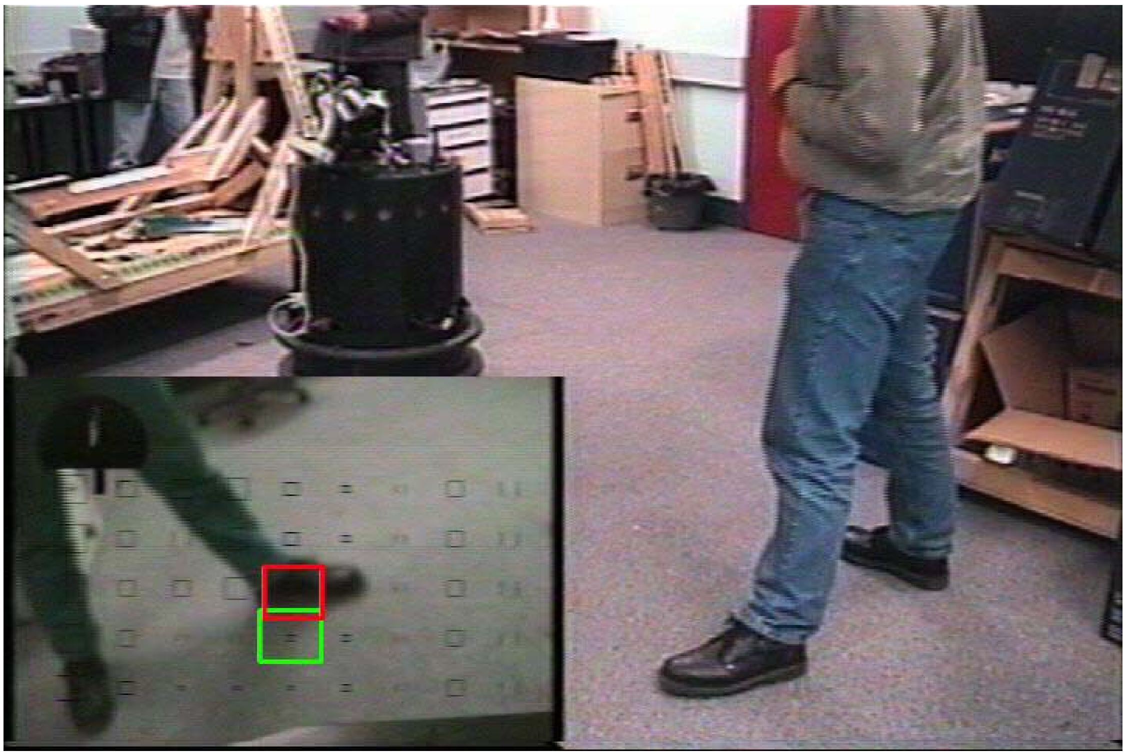


Figure 6.15: View while pursuing a person. Camera mounting is misaligned as in Figure 6.14. The robot's view is inset.

from observing servoing was simple. While the robot was servoing, the system continued to update the carpet match matrix and provided the data, along with the actions executed by the servoing system, to the learning to wander system.

Implementing the learning servoing from observing wandering system was more complicated. While wandering, no interesting targets are being tracked, so there is no data for the servoing system, but if an interesting target is observed the target pursuit behaviour becomes dominant. To allow the servoing system to learn from the actions of the wandering behaviour the behaviour model was modified so that the target pursuit behaviour only becomes dominant for every second interesting target. The first interesting target found while wandering was tracked to provide data to the learning to servo system; when a second interesting target was found target pursuit became dominant. The modification also led to more interesting behaviour because the robot was more likely to wander some distance before servoing. The movie file on the CD-ROM demonstrates the effect of choosing every second target.

As a suggestion for further work, we note that other off-policy learning techniques could also have been applied to the problem. The applicability of *learning from other controllers* is obvious because hard-wired behaviours were already available, from the

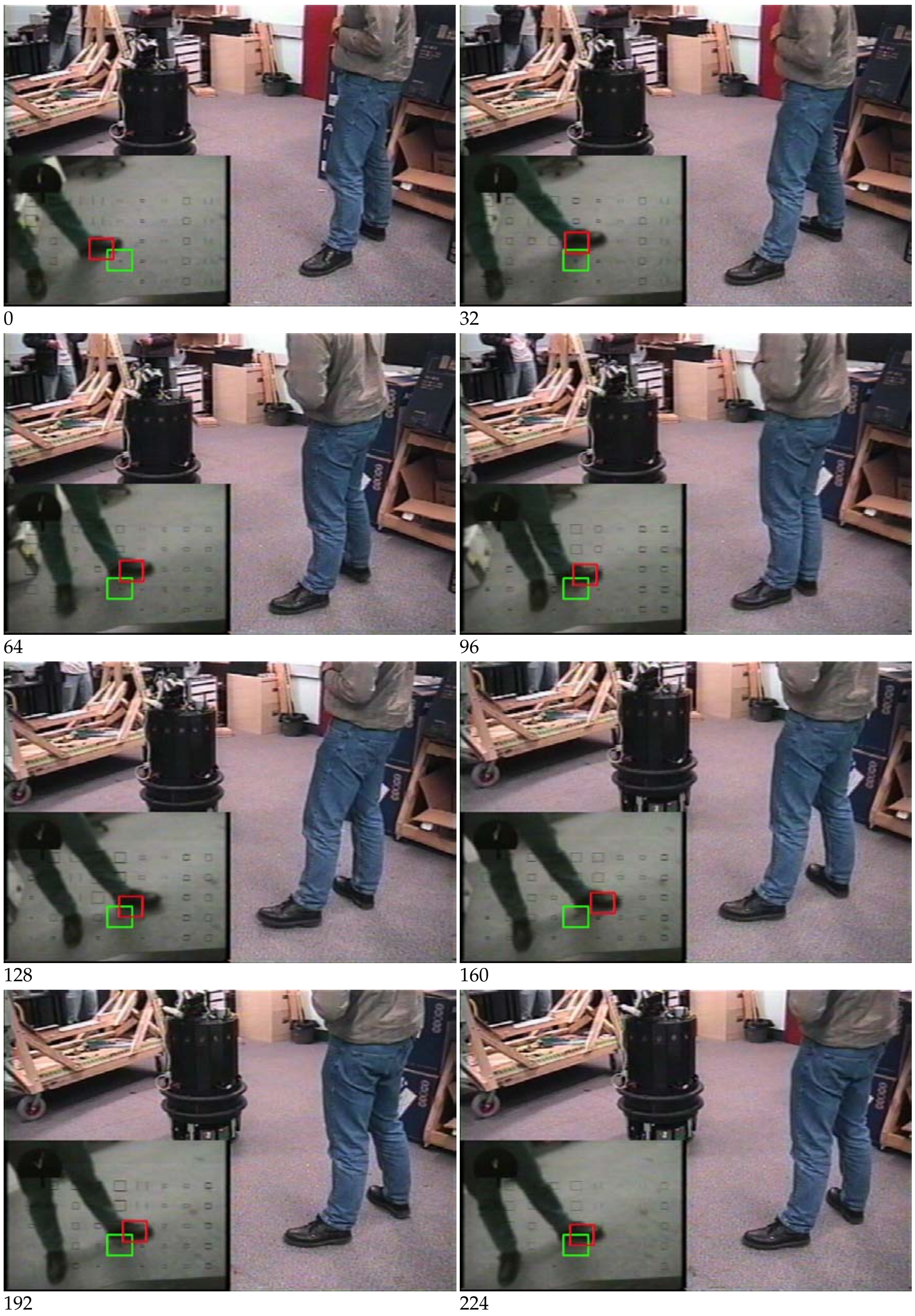


Figure 6.16: Pursuing a person while they take two sideways steps to the left. The target was the person's left foot. The numbers below each image indicate the number of frames at 30Hz; therefore images are around 1 second apart.

work of Cheng and Zelinsky (1996). There are two applications for the *learning from phantom targets* idea to the servoing problem:

1. Learning from other tracked targets: The target tracking system only tracked one target, the current target for servoing. If the system were capable of tracking multiple targets simultaneously the data from all targets could be used for learning to servo.

2. Learning from other goal positions: The servoing task was to move the robot so that the target moves to a central goal position in the robot's view. If the task of the system were broader—to servo the target to varying goal positions—the learning system could consider all possible goal positions while learning.

These ideas can be used together, using many targets and considering them relative to many goal positions.

6.7 Discussion

6.7.1 Learning Difficulties

Early in testing, the velocity of the robot (measured with encoders) was not provided as state information. Learning consistently failed. Analysis of the data showed that because of the restricted acceleration available, the difference between the current and next-state was negligible, being of the same order as the noise in the system. As a result the state information was simply filtered out—there was effectively no correlation between the command and the change in state. Chapter 2 discussed that many implementations of continuous state and action Q-learning did not require that the current velocity be included in the state vector. Past systems also lacked penalties for coarse motion. Without penalties for coarse motion the learned wandering and target pursuit behaviours can shake the robot hard enough to dislodge components of the robot. The WFNN algorithm can produce sudden changes in action by switching between wires.

Adding the velocity to the state representation caused its own intermittent problem. Occasionally a mysterious instantaneous divergence of the learning algorithm occurred. Inspecting the logged state data revealed that just before divergence the robot's system software had returned a velocity from the encoders that approached the speed of light, backwards! This was too much for the WFNN learning algorithm to handle. After this experience we added sanity filters to the state data processing by setting upper and lower limits for measured values. Data outside the range was ignored by both the learning system and the controller.

In retrospect, the exploration strategy of adding normally-distributed noise to the actions was too simple. A more advanced system would have led to faster learning. An example is the exploration system of adding varying magnitude and frequency sinusoidal functions, so that a consistent series of smoothly varying actions is generated (Saito and Fukuda, 1994).

The *learning from other behaviours* approach was an excellent aid to exploration as other behaviours often executed long sequences of consistent motions. The approach was applied in learning from both the learned and hard-wired behaviours. The most common case was to learn servoing by observing the hard-wired wandering behaviour. It is an interesting case since the hard-wired wandering behaviour never moves the robot backwards. The robot does need to move backwards while servoing if the target is too close. Under these conditions the servoing system learnt to servo forwards, then learnt to servo backwards. The result shows that learning from other behaviours was beneficial, but it also shows that the learning system was doing more than just copying pieces of another behaviour.

Learning to wander was more difficult than learning to pursue targets. The wandering behaviour received reward for moving forward. It discovered this quickly, long before any obstacle avoidance skills were learnt. This resulted in an ongoing need for labour-intensive monitoring of the robot. The infrared 'bumper' was not sufficiently reliable to protect the robot or bystanders. The target pursuit behaviour moved more conservatively. It learnt quickly to keep the target in view so movement was minimal. It was time consuming for the wandering behaviour to gather experience of the range of situations it would probably need to face. There are many possible obstacle configurations. Trying to help the robot to learn to wander by rearranging obstacles was counterproductive: the learning algorithm could credit its own actions with causing the change in obstacle configuration! In contrast, it was easy to provide *interesting* objects in various locations as exercises for the target pursuit behaviour. Another factor determining the difficulty of the learning problems was the dimensionality and reliability of the state data. The state space for the wandering behaviour was 37 dimensional—the 5×7 carpet match matrix plus translational and rotational velocity—versus 6 dimensions for the visual servoing system. Also, the carpet match matrix was not as reliable as the target tracking information.

The focus of our research is the learning algorithm—not the vision system—

however, the learning algorithm cannot function with inaccurate or incomplete state data. A non-learning control system can cope with intermittent sensing problems: when the robot moves the sensing problems will often resolve themselves, through a change of viewing angle for example. A learning system cannot separate the false sensor readings from the true and is misled by the experience. This deficiency of learning systems cannot be fully solved through improvements to sensing and processing as it is impossible to sense and model the state of the world exactly.

6.7.2 Vision Difficulties

The free space detection through carpet matching implementation was inadequate for the learning system, even under the assumption that we are only interested in navigating in areas with similar carpet. It does seem to be adequate for the hard-wired wandering behaviour. Only one indicator of similarity to carpet was used, the colour in RGB space. Cheng and Zelinsky's (1996) system had a camera mounted close to the ground and used grey-scale texture. Lorigo et al. (1997) used RGB colour segmentation, but in conjunction with HSV colour segmentation and boundary detection using intensity.

The restricted field of view was a major problem for the wandering behaviour. As the state representation gave only instantaneous information the robot could easily collide with an object that was no longer in view. Another common problem situation was to be caught between two obstacles. In Figure 6.4 a potted plant and a wall are in the robot's view. If the robot turns away from the plant it could pass from view. The robot might then turn away from the wall and end up facing the plant again. The robot would not rely so heavily on its current view if information about the surroundings were stored, even temporarily, and included in the state vector. In the current configuration, however, the state space is already too large.

Automatic target selection through the interest operator could also be problematic. The interest operator sometimes caused unexpected objects to be chosen as targets. Figure 6.17 shows the robot servoing toward a static object when it was expected to pursue a ball. The sequence is also included as a movie file on the CD-ROM. Occasionally targets were tracked that were above floor level (e.g. on a desk). Objects that were fairly uniform in the vertical occasionally cause the template tracking to slide upward. There was a particular problem with selecting a black shoe as a target if a person was wearing black trousers. The shoe was selected as interesting because it was markedly different colour to the carpet. During movement the sock becomes obscured and the template matching process slid the target up and down the person's black trouser leg. The erroneous data was a form of noise for the learning process. Despite the problems

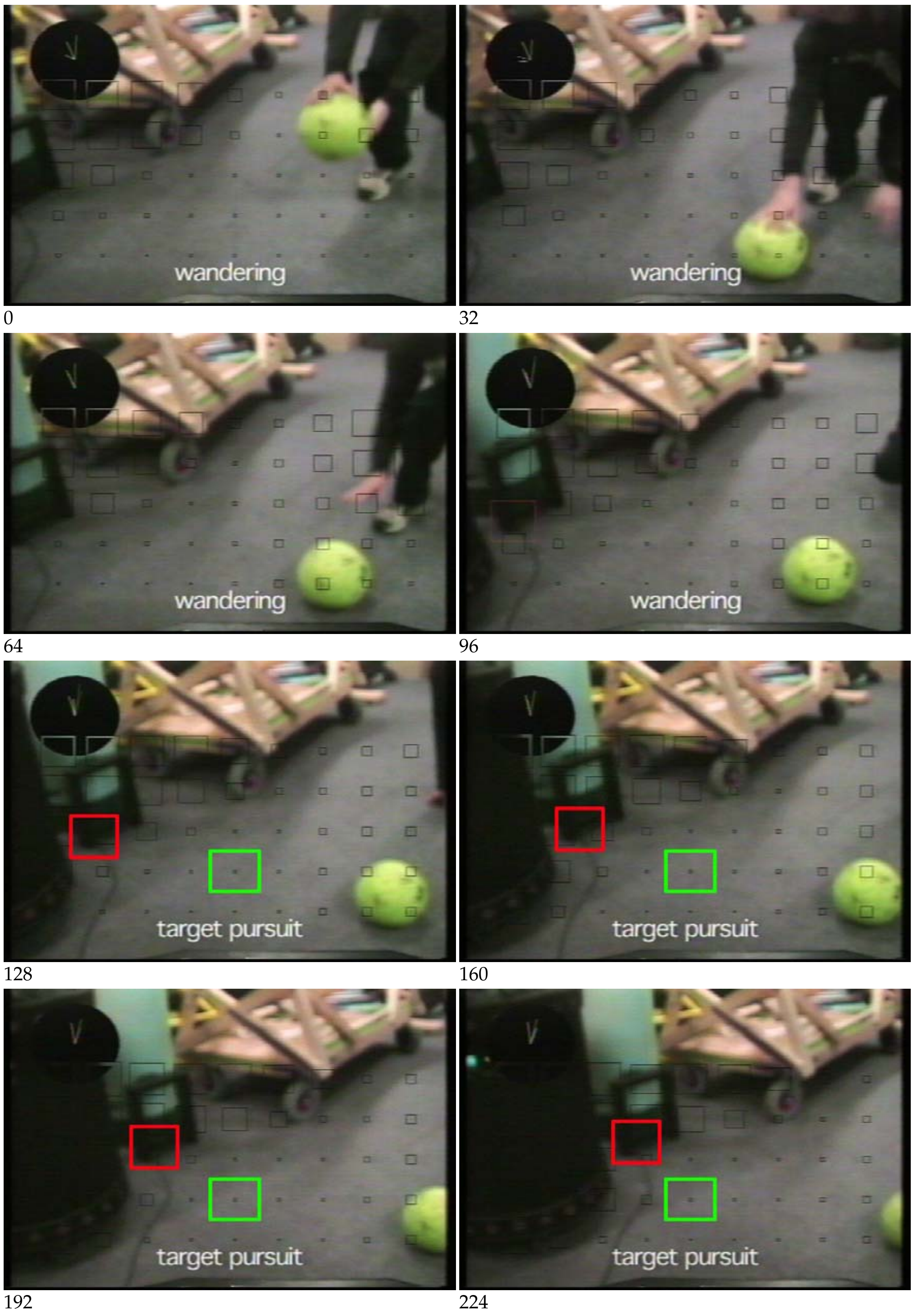


Figure 6.17: The robot was expected to pursue the ball. Instead, the interest operator chose an object leaning against a pillar (*frame 128*). The robot pursued that object. The numbers below each image indicate the number of frames at 30Hz; therefore images are around 1 second apart.

of automatic target selection it did provide amusement when the robot ignored objects intended to be interesting (as shown in Figure 6.17), or fixated unnoticed on a visitor's shoe and began to follow them.

6.7.3 Relation to Other Work

Visual servoing through reinforcement learning with discrete actions was demonstrated by Thrun (1995). Our approach, using continuous states and actions, seems more appropriate for this task as it involves fine positioning and matching velocity with the target.

Learned vision-based wall avoidance for a robot in a constructed environment with discrete actions was described by Terada et al. (1998). The constructed environment was an empty 180cm$\times$90cm rectangular box. Learning in a constructed environment is easier than learning in a more natural environment. The predictability of a constructed environments makes sensory data more reliable. Constructed environments and small robots also reduce safety management problems. Also, the control problem is easier for small robots because they have less momentum.

The Asada Laboratory has developed several vision-based reinforcement learning systems (Asada et al., 1994, 1998). These systems have used discrete actions and adaptively discretised the state space. More recently a visual servoing system using reinforcement learning with continuous states and actions was reported by the Asada Laboratory (Takahashi et al., 1999). Takahashi et al.'s continuous state and action Q-learning algorithm was described in Chapter 2. We noted earlier that their visual servoing system did not include the velocity of the robot in the state representation. This should introduce a perceptual aliasing problem since robot's current velocity is an important factor in determining the motion of the robot in the next time step. Their formulation also did not include any negative reward terms for coarse control. However, the WFNN controller produced coarse control unless we used negative reward terms. We suggest that Takahashi et al.'s algorithm may have produced smooth actions simply because it could not represent coarse actions (Takeda et al., 2000). Rather than being hard-wired, the smoothness of the WFNN controller's actions was produced by learning an optimisation between the different demands of the task.

6.8 Summary

The WFNN method was capable of learning behaviours for a real robot in real-time. The learning system produced novel behaviour and camera calibration was not required. Use of real sensor data and managing the safety of the robot were the major difficulties, as can be expected for any robotics task. The state of the world can never be measured or modelled exactly. For learning algorithms to be practical for robot learning, the period of unsafe robot behaviour must be reduced, and the algorithms must cope with errors in data.

The ability of the WFNN algorithm to learn from off-policy actions was demonstrated. Learning time was decreased through the technique of *learning from other behaviours*. The technique is an easily implemented source of data for learning that leverages the architecture of behaviour based systems.

Adding penalties to the reward function for coarse motions was essential. With these penalties the WFNN algorithm generated actions that varied smoothly with the measured state.

Wandering was learnt through a novel reward function based on the idea that viewing free space and moving forwards are the ingredients for successful wandering. Learning to wander was quite successful. Major impediments were the limited field of view of the camera, and that the system often developed a preference for turning in one direction.

Learning to visually servo was an easier task than learning to wander. The state-action space was easier to explore and the robot was easier to manage. The visual servoing system was also successful when the camera was misaligned, producing zig-zagging motions. This capability is evidence that learning systems can increase the robustness of robots.

Chapter 7

Active Head Experiments

Once a monk drew the master's portrait and showed it to the master.

The master said, "If I look like that, beat me to death. If I don't look like that, burn it."

Joshu, c. 890, no. 502

In the mobile robot experiments described in the previous chapter, the robot's ability to avoid obstacles was hampered by the narrow field of view of its fixed camera. Visual servoing was slow because it required movement of the entire robot. Animals enhance the flexibility of their vision systems through head and eye movements. Active vision mechanisms direct a camera's view through actuation. Active vision heads are active vision mechanisms that imitate animal head and eye movements. This chapter demonstrates learned visual servoing to both static and moving objects using an active vision head. This is the first application of continuous state or continuous state and action reinforcement learning to an active head control problem. The learning system's task was to move the active head's joints so that an object of interest is centred in the view of the left and right cameras. This simple behaviour is the foundation for more complex active vision behaviour. The WFNN-based controller improves its performance based on rewards that are given relative to servo quality.

7.1 Active Vision

The previous chapter discussed purposive vision as a practical replacement for the general vision paradigm. Marr's (1982) pioneering computational vision theory was introduced as an example of the general vision approach. This chapter continues the discussion, focusing on the link between vision and movement. Marr's work mentioned movement but treats it as a passive process of observing moving objects, or observing while you happen to be moving; eye movements are treated as something to be edited out, rather than an essential part of the process of seeing (Marr, 1982, pp. 51, 159, 345). This can be partly explained by the level of technology available at the time: processing images in real-time was impossible, therefore image processing research concentrated on pre-recorded images.

Human perception research was also influenced by the availability of technology: the invention of the tachistoscope in 1850 led to a concentration of research on the unmoving eye and static objects that continued until various eye movement measuring devices were developed in the 1890s (Heller, 1988). More recent human perception research emphasises the strong relationship between eye movement and perception (Carpenter, 1991b). Traces of eye movement show that people move their eyes repeatedly so that different parts of the scene map onto the *fovea*, the region of highest acuity near the centre of the field of view (Yarbus, 1967). In addition to these movements, human eyes make small unconscious movements constantly. Objects that remain stationary relative to the retina for a few seconds fade from view (Yarbus, 1967).

Animals use eye, head, and body movements to obtain different views. Some animals judge distance through parallax generated by head and body movement, others use the difference in view between the left and right eyes. Many animals have more or less independent control of each eye, but some have less eye movement range, accommodation (variable focus), or degrees of freedom than humans (Tansley, 1965). Humans do not have independent eye control. Control is yoked so that the eyes gaze in the same direction: the eyes roll up or down together, and pan to the sides with synchronised *vergence* and *versional* movements (Judge, 1991). In vergence movements, *convergence* and *divergence*, both eyes turn inward or outward to look at an object at a particular distance. Vergence coincides with *accommodation* (focusing). In versional movements both eyes turn left or right to focus on an objects lateral position.

Basic human eye movements include *saccades, smooth pursuit*, and the *vestibular-ocular reflexes*. In general, the purpose of these movements is to allow the gaze to *fixate* on a particular object. Saccades are jerky eye movements that are generated by a change of focus of attention (Becker, 1991). During these fast eye movements the target is not visible, making saccades an *open-loop* mechanism.

Smooth pursuit movements follow moving objects and keep them in the centre of the field of view (Pola and Wyatt, 1991). The eye movement is driven by the movement of the object, not its position. Positional errors are corrected through saccades. Smooth pursuit is a *closed-loop* mechanism.

The vestibulo-ocular reflex (*VOR*) compensates for head movement through inertial measurements made by the semi circular canals (Jones, 1991). The *optokinetic* reflex (*OKR*) compliments VOR by measuring movement of the scene (Collewijn, 1991). These are also closed-loop mechanisms.

Despite the links between vision and action in animals, development of most early active vision mechanisms was motivated by the 3D reconstruction and general vision paradigm rather than biological parallels (Moravec, 1990; Aloimonos et al., 1988). Obtaining multiple views through actuation assisted the general recovery process. Ballard and Brown (1992) described the benefits of gaze control, as opposed to static, multiple views:

1. Multi-resolution vision: sensors with higher acuity in the centre, or artificial fovea, need gaze control to centre an object of interest in the higher acuity area.

2. Motion blur: objects that are moving relative to the camera view become blurred. A gaze control system can eliminate motion blur by keeping pace with a moving object. This also causes the background to become blurred, easing the problem of segmenting an object of interest from the background.

3. Mathematical simplification: fixating objects in the centre of the field of view simplifies the projection model.

4. Facilitating stereo fusion: fixating objects in the centre of the field of view zeroes stereo disparity at the fixation point.

5. Expanding the field of view: passive lenses and mirrors can also expand the field of view, but at the cost of lower resolution.

Working with active vision systems led to the recognition that acting and sensing interact strongly; vision cannot be performed in isolation from the task and the mechanisms. The concept of *purposive* or *animate* vision was proposed based on these ideas (Bajcsy and Allen, 1984; Burt, 1988; Aloimonos, 1990; Ballard, 1991; Ballard and Brown, 1992). Active vision systems acquire visual data selectively in space and time. They exhibit the ability to focus attention on relevant information. A purposive active vision system senses to move, and moves to sense, in a way that is directed toward performing some task.

7.2 Learned Gaze Control

Bajcsy and Campos (1992) and Ballard and Brown (1992) both proposed that the ability to learn is a necessary ingredient for an active vision system. Gaze control is a prerequisite ability for realising higher level benefits of active vision. Investigation of biological systems has shown that adaption and learning improves the efficiency of gaze control and compensates for imperfections or changes in the eyes.

Leigh and Zee (1991) surveyed oculomotor disorders and described numerous adaptive mechanisms that partially compensate for deficiencies. Weak eye muscles result in stronger nerve impulses being sent to compensate. Seeing through spectacles requires an adaption to the optokinetic reflex gain due to the prismatic effect of the spectacles; adjustment to new spectacles takes only minutes (Collewijn et al., 1983). Control of saccadic movements can also adapt to the presence of lenses or prisms (Becker, 1991).

Human infants develop eye movement skills incrementally (Aslin, 1981). Infants make many small, equally-sized saccades to fixate on a target. They develop towards the adult model of making only two or three movements: the first movement covers around 90% of the angular distance; remaining movements correct the error. Smooth pursuit skills also develop slowly. Although there are explanations other than adaption for the incremental development, such as infant sensory deficiencies, the adult ability to adapt saccadic movements when exposed to distorting prisms provides some evidence for adaption.

Adult human saccadic and smooth pursuit mechanisms increase efficiency by adaptively predicting the motion of the target (see for example Jürgens et al., 1988; Zhao et al., 1993). Prediction helps to compensate for the delay introduced by the vision processing system.

The diversity of adaptive oculomotor mechanisms in biology supports the proposal that artificial learning systems are important for active vision systems. The following sections survey existing work in learned gaze control for active vision systems.

7.2.1 Learned Smooth Pursuit

Berthouze et al. (1996) demonstrated learned smooth pursuit using Feedback Error Learning (FEL) (Kawato et al., 1987). The active head learnt to move so that a moving target stayed in the centre of the field of view. In the FEL architecture, an inverse dynamic model is learnt incrementally in order to compensate for an underlying controllers error in following a specified state trajectory. The learned inverse dynamic model is a feedforward component of the combined controller. Accordingly, Berthouze et al.'s learning controller was built on top of a simple feedback controller.

Berthouze et al.'s research assumed independence of the degrees of freedom and implemented separate controllers for each joint to simplify the problem. The tilt joint was controlled to minimise the vertical error. Left and right rotation joints were controlled independently to minimise the horizontal error in their respective images. The pan joint was controlled to minimise the difference between the horizontal position of the target in the left and right camera images. This removed the problem of redundancy between the left and right rotation and pan joints. The system learnt to improve on the simple hard-wired controller within 20 seconds. The system ran at 30Hz.

The formulation of FEL used in Berthouze et al.'s experiments did not allow the use of arbitrary cost functions or optimisation of performance over time. Consequently, there is a delay in the tracking since the system cannot compensate for the 50ms delay introduced by image acquisition and processing. Despite the delay, tracking error was less than 2 degrees of joint angle for a swinging target with a period of around 7 seconds. The magnitude of the swinging motion was not reported. The current velocity of the target in terms of pixels was not an input to the controller. Therefore, the controller could not compensate for the target velocity; it could not make a different decision based on whether the target is swinging towards the centre of the view or away.

7.2.2 Learned Vestibulo-Ocular Reflex

Shibata and Schaal (2001a) developed a model of the vestibulo-ocular reflex (VOR) (Jones, 1991) and optokinetic reflex. The VOR model included a learning component. They demonstrated the model by controlling head pan to stabilise the viewed image while the robot's body moves the head. The vestibular signals were produced by a gyrosensor. Like Berthouze et al. (1996), Shibata and Schaal used Feedback Error Learning. However, Shibata and Schaal extended the FEL framework to cope with some delay by adding an eligibility trace mechanism (Barto et al., 1983). The VOR model included velocity components as well as position components.

Shibata and Schaal's system learnt quickly; they reported excellent performance after only 30 to 40 seconds of learning. The fast learning is partly due to the hard-wired underlying model that implements a rough VOR reflex even before learning.

7.2.3 Learned Saccadic Motion

Learned saccadic motion through artificial ocular-motor maps has been demonstrated by several researchers (Ritter et al., 1992; Rao and Ballard, 1994; Marjanović et al., 1996; Pagel et al., 1998). The ocular-motor map forms a mapping between visual coordinates and joint angles. Providing a target's position in visual coordinates as input to the map produces a joint position that should bring the target into the centre of the field of view. If executing the saccadic action does not bring the target into the centre of the field of view the ocular-motor map is updated. The ocular-motor map is usually initialised to a linear mapping. Updates to the map gradually improve saccade accuracy. The ocular-motor map technique is especially applicable to acuity variant camera systems which have non-linear mappings between joint angles and visual coordinates. Acuity variant systems are difficult to calibrate through more conventional approaches. Non-uniform approximation resource allocation is important for ocular-motor maps since higher mapping accuracy is required around the centre of the field of view (Pagel et al., 1998).

Current ocular-motor map techniques are based on target position in the image; they do not consider target velocity. Human saccades are based on both position and velocity so that the saccade can compensate for the expected movement of the target. Estimating the target velocity before the saccade is generated also allows smooth pursuit to commence immediately at approximately the correct velocity (Yarbus, 1967). Murray et al. (1995) demonstrated the importance of considering image velocity for saccades with a non-learning controller. An ocular-motor map that includes the image velocity would be a significant advance on current methods.

Existing ocular-motor map approaches to saccadic motion further simplify the model of human saccadic eye motion by providing an actuator that has direct control over the camera angle. In effect they add an implicit feedback-loop that ensures that the camera angle matches the output from the ocular-motor map. After the movement the new view is processed to determine if another saccade is required. Human saccadic eye movements are generated by a nerve impulse that approximately rotates the eye to the desired position. Corrective saccades are often generated based on the error in the movement, rather than an error assessed through vision. This has been demonstrated by removing the visual target of a saccadic motion while a saccade is in progress: a corrective saccade is generated without a visual target (Becker, 1991). Implementing an open-loop approach to saccadic motion would be an interesting extension of current methods.

7.2.4 Babybot: Learned Saccades, Smooth Pursuit, and VOR/OKR

Researchers at the LiraLab followed a developmental approach in adding skills to their active head and arm system called Babybot (Metta et al., 2000). They demonstrated learned smooth pursuit, saccades, vestibulo-ocular and optokinetic reflexes, and predictive neck movement. Learning was also employed to integrate the head and arm systems. The learned components formed an integrated system that pursued objects, combining smooth and saccadic movements with compensation by the vestibulo-ocular system.

The learned smooth pursuit system incrementally identified the image to motor Jacobian. This is a standard approach for image-based visual servoing (Hutchinson et al., 1996). Metta et al. multiplied the error in image coordinates by the inverse of the Jacobian by a small negative scalar (found empirically) to obtain the error in motor coordinates. As with Berthouze et al.'s (1996) approach, Metta et al.'s smooth pursuit controller did not consider the image velocity of the target, treating it only as a source of error in the learning process. It is notable considering the human vision motivation for both projects: human smooth pursuit is mostly driven by image velocity (*retinal slip*) (Pola and Wyatt, 1991). Murray et al. (1995) demonstrated successful velocity and hybrid velocity/position smooth pursuit with a non-learning controller. In human vision, the transition between saccadic and smooth motion sometimes includes a creeping corrective motion, or *glissade* (Becker, 1991). Glissades may reduce the need for position information in the human smooth pursuit system.

Metta et al.'s artificial saccadic system was based on a learned ocular-motor map similar to the saccadic systems described earlier. Again, velocity of the target was not considered. The neck (pan) controller minimised version so that the head pointed toward the target. The controller reduced lag by using the predicted destination of saccadic eye movements.

Panerai et al. (2000) used the LiraLab active head's artificial vestibular system to compensate for induced pan movements, imitating the vestibulo-ocular and optokinetic reflexes (Jones, 1991; Collewijn, 1991). The inputs to the learning system were the image velocity (retinal slip) and pan velocity, as measured by the vestibular sensor. The output from the learning system was a camera angular velocity command. The neural network learnt to minimise the image velocity by modifying its output relative to the retinal slip under the assumption that the sign of the required movement is known.

7.2.5 Learned Vergence

Learned vergence motion through reinforcement learning was demonstrated by Piater et al. (1999). The task was to control the vergence angle of a binocular head to minimise disparity between the views of the cameras. Five discrete actions were available: changes in vergence angle between 0.1 and 5 degrees. The direction of the change was hard-wired. States were also represented discretely, including the positioning error. The state representation included the number of movements allowed before the end of the trial (maximum five) and an indicator of the uncertainty in the position of the target. The representation allowed the learning system to make a trade-off for final accuracy. The pure-delayed reward signal was the negative of the final positioning error. Given that the error signal is available at all times a delayed reward statement of the problem would probably result in faster learning. The choice of a discrete state and action representation without generalisation is ill-suited for the task. Vergence motions were made using discrete actions in a few steps, in contrast human vergence motions are smooth and closed-loop (Judge, 1991). The lack of generalisation in the state and action representations requires exploration of every possible representable state and action. Piater et al.'s work appears to be the only application of reinforcement learning to active head control. Its main limitations are the simplicity of the task, given that the direction of movement was hard-wired; and the discrete state and action, open-loop, pure-delayed reward formulation of the problem.

7.2.6 Discussion

From a learning point of view, an active head is an excellent platform for experiments since it has configurable degrees of freedom, fast dynamics, relatively noise free state data, and is less dangerous than a mobile robot. A continuous state and action learning algorithm is appropriate for this control problem; to date, however, reinforcement learning with continuous states or continuous state and action has not been applied to the problem. From the active vision point of view, learning can be justified by the presence of various adaptive mechanisms in human vision, the unpredictability of the environment, and uncertainty in the active head's mechanical and optical parameters. Another motivation for use of an active head in this thesis is that demonstrating a general-purpose algorithm using only a single robot is unconvincing.

The learning task in this chapter is visual servoing. Visual servoing is smooth, closed-loop control, thus it could be seen as similar to the human smooth pursuit system. A major difference is that the human smooth pursuit system is driven by the velocity of the object of interest, not its position. In contrast, visual servoing can be driven by the position (and optionally velocity) of the target. Human saccades are driven by the position and velocity of the target but are open-loop. Because of these differences, the learning task in this chapter is not an attempt to duplicate human gaze control mechanisms. The main focus is to overcome the limitations of existing learning gaze controllers by coping with delays, producing smooth movement, and compensating for target velocity.

7.3 Experimental Platform: HyDrA

The experiments in this chapter were performed on the HyDrA binocular active vision platform shown in Figure 7.1 (Sutherland, Rougeaux, Abdallah, and Zelinsky, 2000a). HyDrA has four mechanical degrees of freedom as shown in Figure 7.2. The *pan* joint turns the entire head from side to side. The *tilt* joint sets the elevation of a platform that is common to both cameras. The left and right cameras are rotated independently around the verge axis by the *left and right verge-axis joints*. HyDrA follows the Helmholtz or common elevation configuration since the tilt joint operates before the left and right verge-axis joints (Murray et al., 1992; von Helmholtz, 1962).

HyDrA's pan and tilt joints are actuated through cable drive. The pan and tilt motors are located in the base of the assembly; high-tensile cables link the motors to the joints. The arrangement avoids the need for the tilt motor assembly to be moved by the pan joint motor. Cable drive eliminates backlash and slippage, requires no lubrication, and provides high transmission efficiency (Sutherland et al., 2000a). The use of cable drive for vision mechanisms has an illustrious ancestry in the cable-driven eye-movement simulators (*ophthalmotropes*) of Ruette, Wundt, Knapp and others in the 1850s and 1860s (Heller, 1988; von Helmholtz, 1962). Examples of this work are shown in Figures 7.3 and 7.4. HyDrA was developed based on success with a cable driven monocular design (Brooks et al., 1997). HyDrA's left and right verge-axis joints are driven by harmonic drive motors rather than cable drive; the name *HyDrA* is an acronym for ***Hy***brid ***Dr***ive ***A***ctive head.

HyDrA is equipped with a pair of colour NTSC cameras. The cameras have a narrow field of view and do not have fovea. Saccadic motion is not useful for this configuration. Camera rotation around the tilt and verge axes imitates human eye movement. Unlike human eye movement, some parallax occurs because the optical axis is not exactly

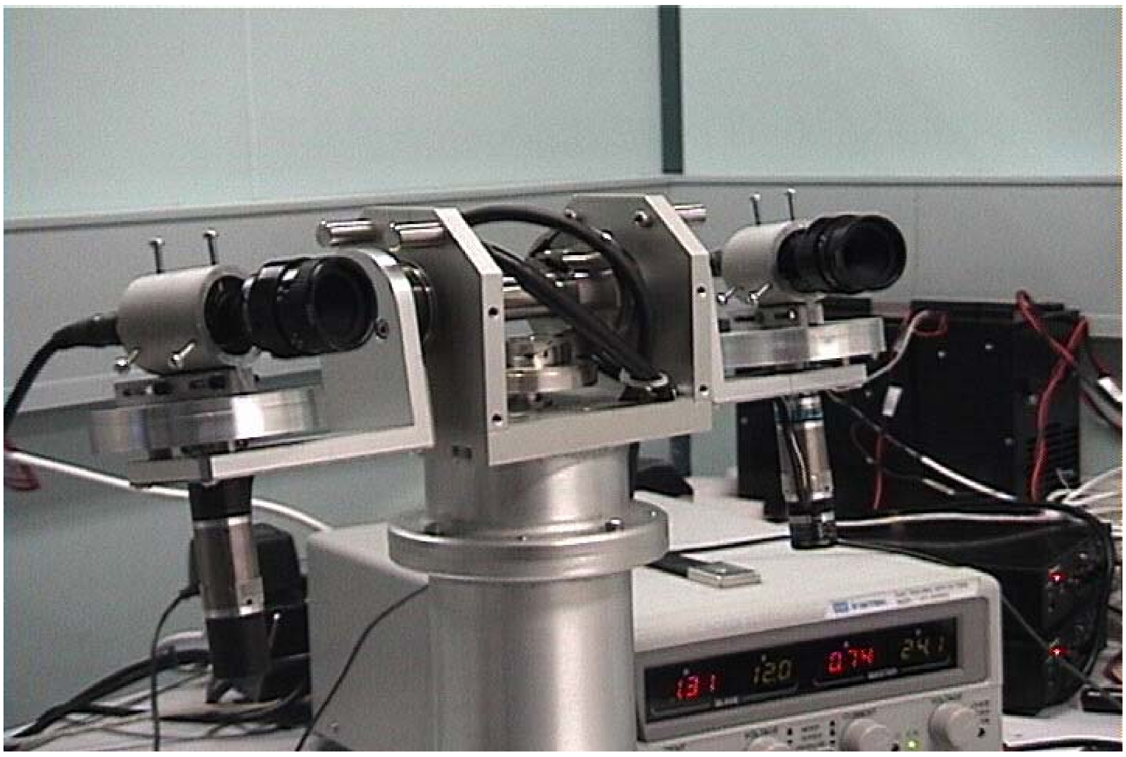


Figure 7.1: HyDrA

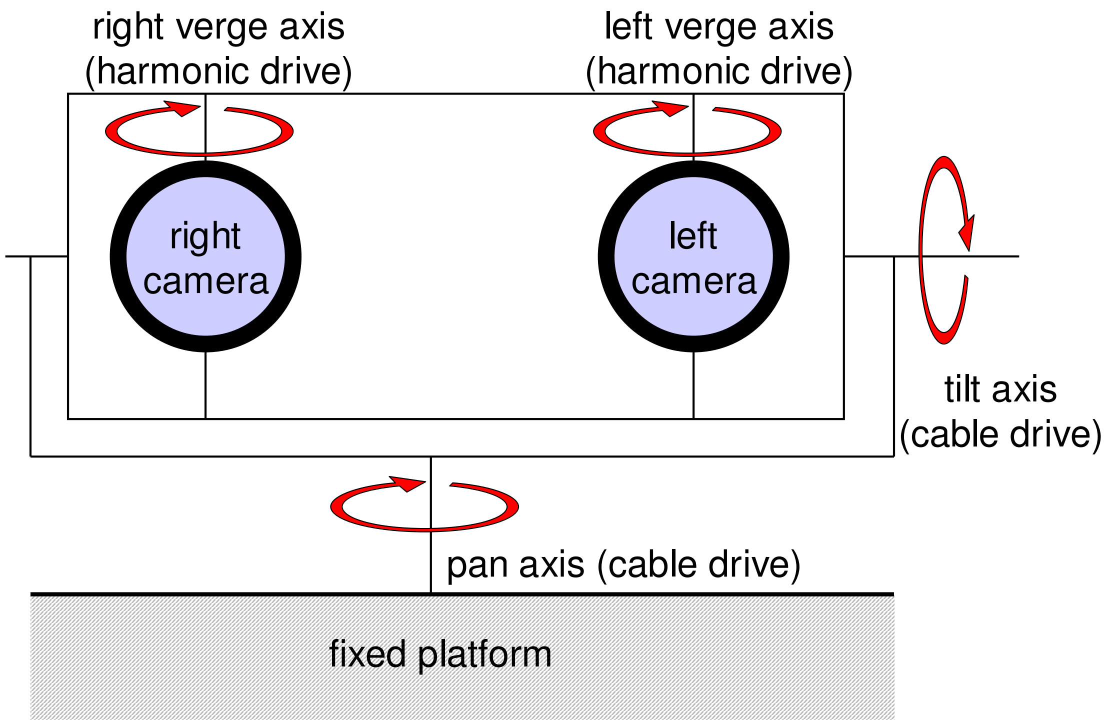


Figure 7.2: HyDrA's degrees of freedom

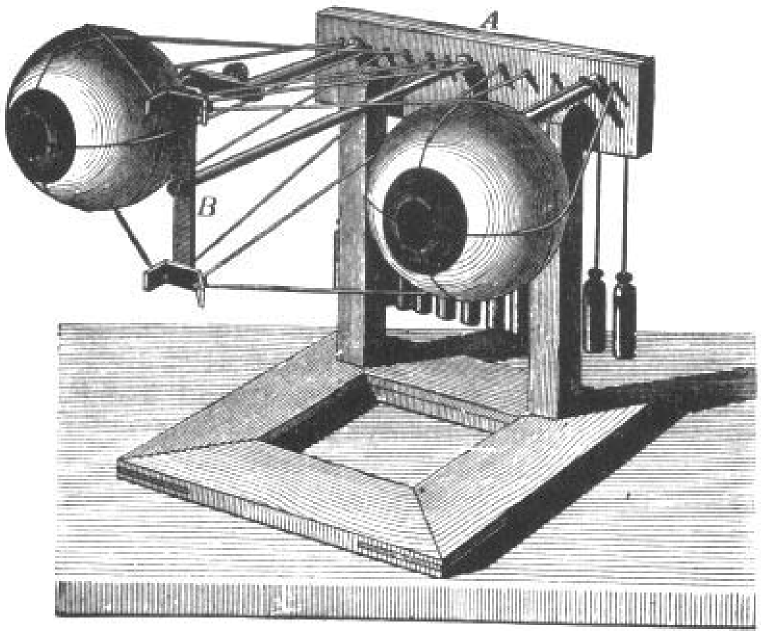


Figure 7.3: Binocular opthalmotrope designed by Ruete (1857).
Source of illustration: von Helmholtz (1962).

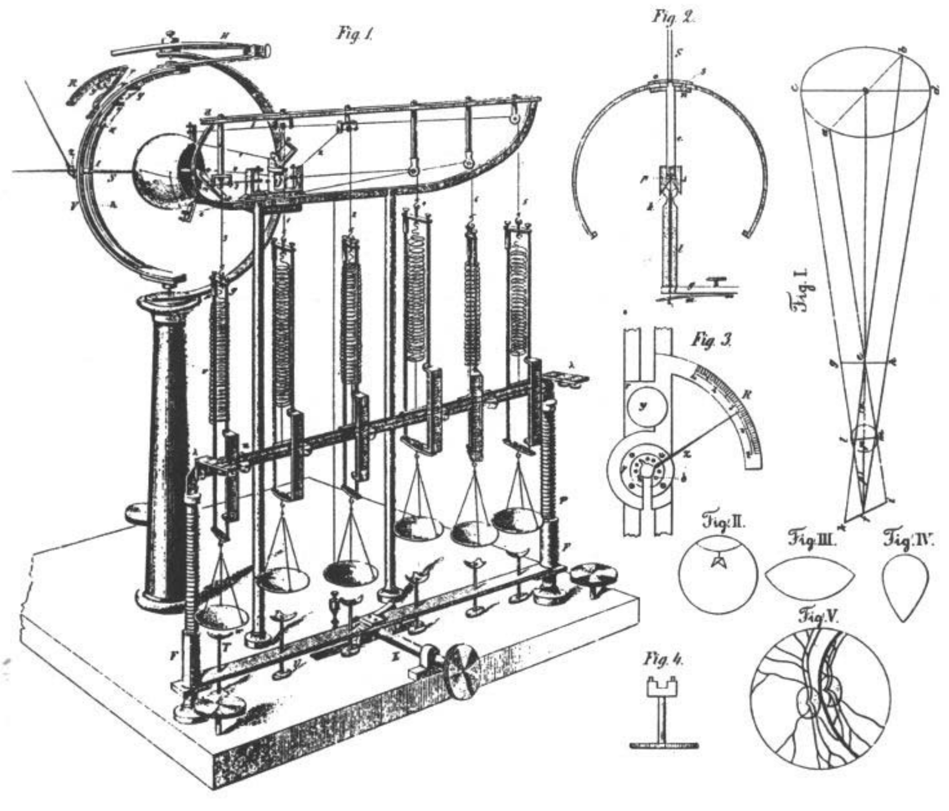


Figure 7.4: Monocular opthalmotrope designed by Wundt (1862).
Source of illustration: Heller (1988).

aligned with the rotational axis. The misalignment is not part of the original design; it is the result of inevitable wear and adjustments to the cameras. HyDrA's pan axis imitates human neck rotation. The pan joint is slower than the tilt or verge-axis joints. HyDrA has less degrees of freedom than the human neck, omitting sideways (roll) and front to back tilt. Neck tilt could be considered to be redundant since HyDrA has eye-like tilt; the difference is that human neck tilt causes parallax since human eyes are frontally located. HyDrA's pan joint also causes parallax but is not equivalent to human neck movement since human eyes are located forward of the neck rotation axis. HyDrA's cameras are set to a fixed focus and fixed iris; there is no imitation of human accommodation mechanisms.

It can be convenient to control the left and right verge-axis joints in terms of vergence and version. In human vision research this approximation to the human coupling scheme is known as Herring's law (Judge, 1991). HyDrA's left and right verge-axis joints can be controlled independently. Any configuration of left and right verge-axis angle can be specified in terms of vergence and version. But if the joints are being controlled independently, and the cameras are possibly pointing at different targets, it seems inappropriate to use the terms vergence and version to specify the left and right rotation around the version axis. The terms *left vergence*, *right vergence*, and *independent vergence* that are occasionally used in the active vision mechanisms literature also seem inappropriate, since vergence is a synchronised activity that always changes the angle between the lines of sight of two eyes (Judge, 1991). Instead, we use the terms *left* or *right verge-axis rotation*.

7.4 Experimental Task

The aim of the experiments described in this chapter was to learn a controller for HyDrA using the WFNN algorithm. No attempt was made to directly imitate the human vision system's control scheme of saccades and smooth pursuit; the only goal was to fixate on static and moving targets. The dynamics of the system and the camera parameters were unknown to the controller. This is the visual servoing task as in the previous chapter. The main differences are due to HyDrA's configurable degrees of freedom and faster movement.

The tracking target used in these experiments was a lead weight suspended with string. The string was replaced with elastic when movement in three dimensions was required. Elastic allows interesting movement without continuous human intervention. The experimental configuration is shown in Figure 7.5.

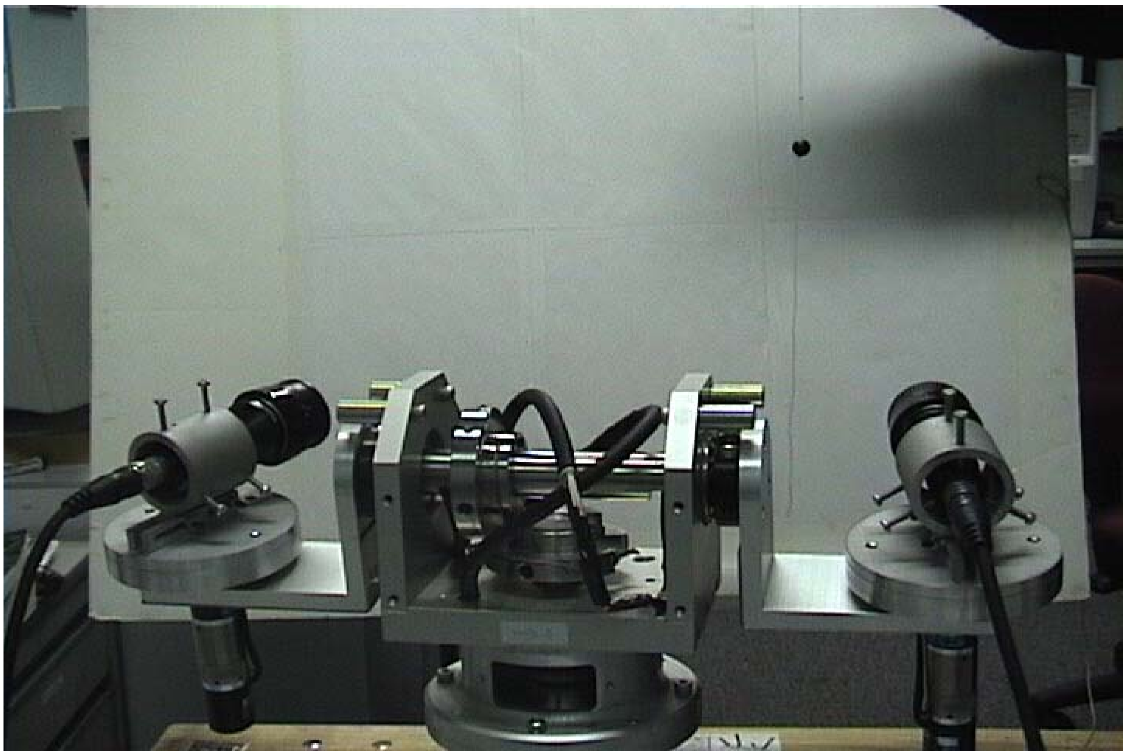


Figure 7.5: Experimental apparatus

Object tracking was performed independently for both of the monochrome cameras through grey-scale template matching. The independent tracking could cause the cameras to fixate on two different objects. To reduce the problem a white background was used. This was reasonable since the focus of these experiments is learning, not vision research. Another image processing algorithm could easily be used with our learning algorithm. If the target was lost in either camera's view then the state data was considered invalid for the purposes of learning. As described in Chapter 5, the learning system controlled HyDrA in these situations without learning from the invalid state data.

A single learning controller controlled the velocity of all active joints. The computational load was divided between two computers using the *act-learn-think-C/S* architecture described in Chapter 5. No independence assumption was made: manipulating any joint can affect any other joint, and either image axis on either camera. This increases the complexity of the controller but also increases flexibility: the controller is suitable for parallel mechanism active heads that have coupling between the joints, for example CeDAR (Truong, 2000; Sutherland, Truong, Rougeaux, and Zelinsky, 2000b), or mechanisms in which the camera axes are not aligned with rotation axes. Independent control of the verge-axis joints allows interesting non-human like behaviour, for example searching for a target with each camera independently. The controller's goal was to fixate on the visual target; there were no specific sub-goals for individual joints. This introduces a redundancy between the pan and verge-axis joints that must be resolved by the learning system. Previous learned gaze control projects defined a separate control task for the pan joint to remove the redundancy (Berthouze et al., 1996; Metta et al., 2000).

7.4.1 Task Representation

In order to approach the visual servoing problem through reinforcement learning a state, action, and reward formulation must be chosen. The formulation for the visual servoing problem in the previous chapter is suitable, but needs to be expanded for configurable degrees of freedom.

HyDrA's configurable degrees of freedom were exploited through a flexible software architecture that allowed any combination of HyDrA's joints and visual axes to be included in the learning problem. The flexibility allowed the complexity of the learning problem to be adjusted and the problem of redundancy to be investigated.

Figure 7.6 gives the reinforcement learning specification. Each included joint (pan, tilt, left verge-axis, right verge-axis) adds 2 state variables to the state representation: joint angle, and joint velocity. Joint angular velocity as returned by the HyDrA system software is a copy of the last velocity command sent; it is not measured. The joint position was included because the joint behaves differently at the extremes of its range of rotation. The number of action variables was the number of joints included. Included joints also add coarseness penalties to the reward function.

The 4 visual axes (left camera x axis, y axis; right camera x axis, y axis) could also be included in any combination. Each visual axis added 2 state variables: target position error in pixels, and target velocity (change in pixels); and corresponding reward penalties for target image position and velocity. These parameters are shown in Figure 7.7. It was appropriate to include the image velocity in the reward function as the estimation of the image velocity was more accurate than previously. Image velocities were estimated from the position of the target in two consecutive frames.

We found that using the square of error terms in the reward function, as in the mobile robot experiments in Chapter 6, led to a controller that fixated on the general area of the target, rather than fixating accurately. Using the absolute value of the error terms, as shown in Figure 7.6, produces higher fixation accuracy. The option to include any combination of the 4 joints and 4 visual axes allowed $2^{(4+4)} = 256$ configurations. Two sample configurations are shown in Figure 7.8. The figure also shows the desired be-

State For each included joint:
  θ: joint angle
  $\dot{\theta}$: joint angular velocity

For each included image axis:
  x: pixels error to target
  Δx: pixels/second velocity of target

Action For each included joint:
  $\dot{\theta}'$: desired joint angular velocity

Reward For each included joint:
  $-\left|\dot{\theta} - \dot{\theta}'\right|$: smooth joint motion

For each included image axis:
  $-|x|$: movement to target
  $-|\Delta x|$: keep target still

Figure 7.6: Reinforcement learning specification for the active vision experiments

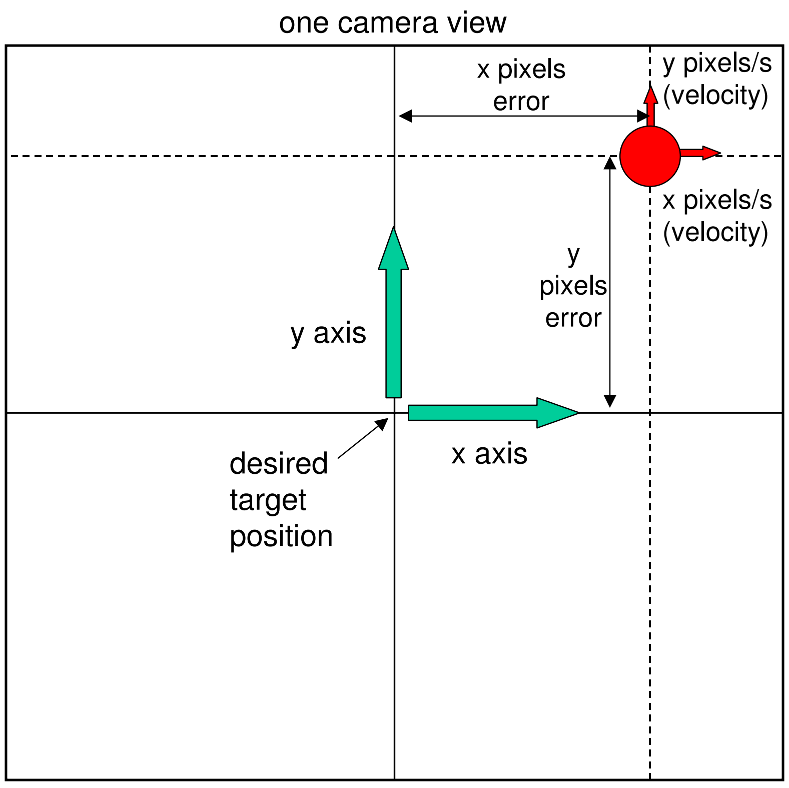
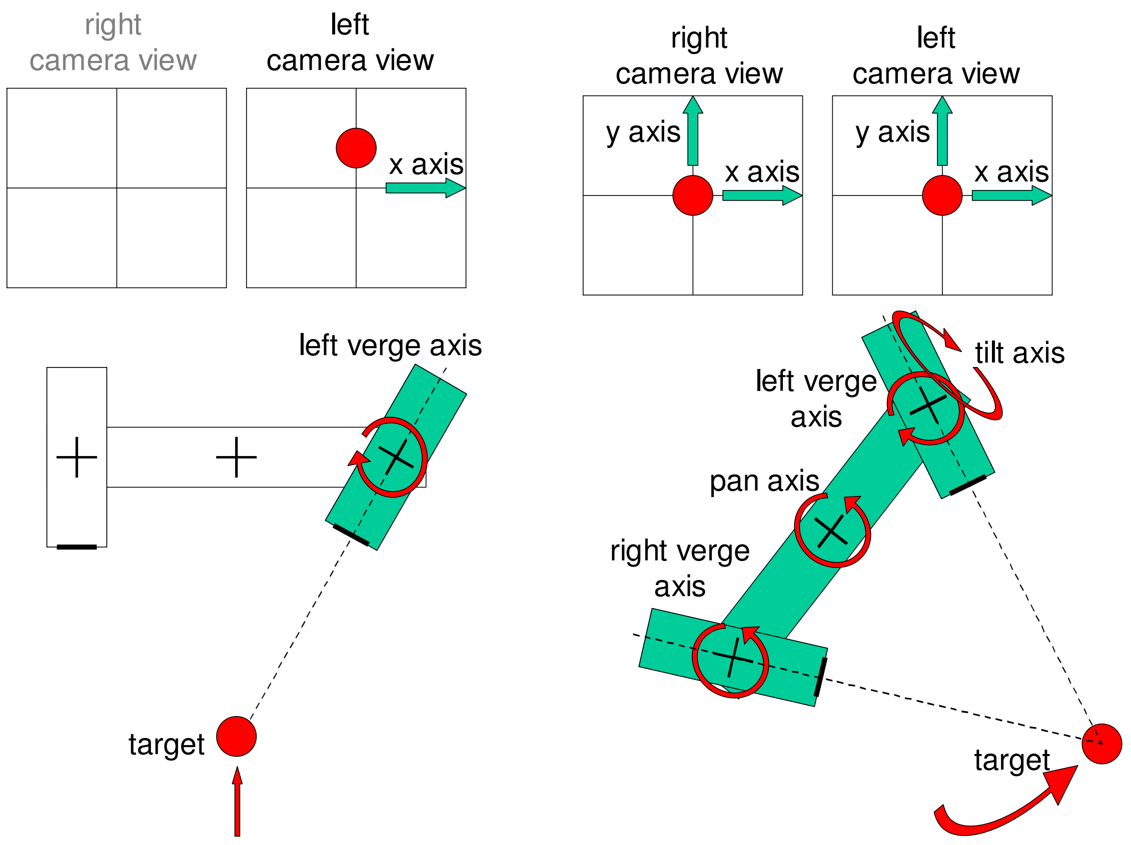


Figure 7.7: Information taken from the camera view for state and reward calculation

haviour of the controller. In the simplest configuration, where only 1 joint and 1 camera axis are included, there are 4 state variables and 1 action variable; the scalar reward will have 4 components. In the most complex configuration, where all 4 joints and all 4 camera axes are included, there are 16 state variables and 4 action variables; the scalar reward will have 16 components.

All of the state and action variables in the control task are continuous. Consequently, it is an ideal task for continuous state and action learning. Fine discretisation would lead to an enormous state-action space—there is no simulator available to synthesise the enormous number of experiences required to estimate so many action-values. Coarse discretisation would lead to coarse control. Rather, we desire accurate, smooth motion and high sensitivity to target movements.



(a) 1 joint, 1 visual axis

(b) 4 joints, 4 visual axes

Figure 7.8: Example joint and camera axis inclusion configurations. Green fill indicates included joints and camera axes. The red arrows are an example of target movement and corresponding joint rotations.

7.5 Single Degree of Freedom Visual Servoing

In the experiments described in this section the WFNN system controlled the right verge-axis joint to fixate the right camera on the target in the horizontal (x) axis. Since the vertical (y) axis was not relevant, the target was suspended with string rather than elastic.

7.5.1 Learning Process

The experiment began with target image template acquisition. With the active head's motors disengaged, the camera was pointed towards the target for template capture. In practice it was not necessary to make a new target template for every run since the target was simple and the background was plain.

The active head system software's start-up sequence then began, finding the physical limits of movement of all joints, and finally positioning the head so that it faced forwards with the cameras slightly verged. The target was positioned manually so that it was visible to the camera.

The learning system was then engaged. Actions were executed at 15 Hz. Initial joint movements were small and the stationary target remained in view, but as movements increased the target often passed out of view of the camera. The target must then be moved back into the camera's view for further learning to occur. If the target is continuously moved manually so that is always in the centre of the field of view the learning system fails because the relationship between joint movement and target movement has been lost: all joint movements appear equally good as the target remains in the centre of the field of view and rewards are consistently high. A delicate compromise must be made between constantly losing the target—thus not learning at all; and constantly moving the target manually—thus invalidating the learning process.

Once the controller was dealing with the static target reasonably well the difficulty of the task was increased, firstly by moving the target slowly, then through faster movements and step (instantaneous) movements. Step movements were performed by covering the target, moving it, then revealing the target. The instantaneous movement did not interfere with the learning process since tracking fails while the target is covered: as described earlier, the learning system does not learn from state data gathered when the target is not visible. Otherwise, the actions that the learning system performed just before the target moved would erroneously change in value based on the new position of the target. When the target was covered, then revealed, the new position of the target was clearly beyond the learning system's control, and it was not expected to guess the new position.

During learning it is important that the learning system experience a wide variety of states. Most importantly, the target object must be seen in many locations of the visual field. Providing the experience was labour-intensive. To ease the burden we constructed a training apparatus.

7.5.2 The Training Apparatus

The training apparatus is shown in Figure 7.9. The purpose of the device is to generate a mixture of smooth and step like target motions. The target was a suspended lead weight, as before. Several other targets were fixed to the white backing-board, lower than the swinging target. A white cylinder hung below the lead weight. The cylinder obscured the static targets one by one as the lead weight swung past.

The training apparatus functioned as follows: At system initialisation, several targets were visible to the camera. The target with the highest correlation with the stored template image was chosen automatically as the *current target*. This can be regarded as a random selection. Since the target tracking process searched for the current target in a small window around the current target's last location it remained fixated on one

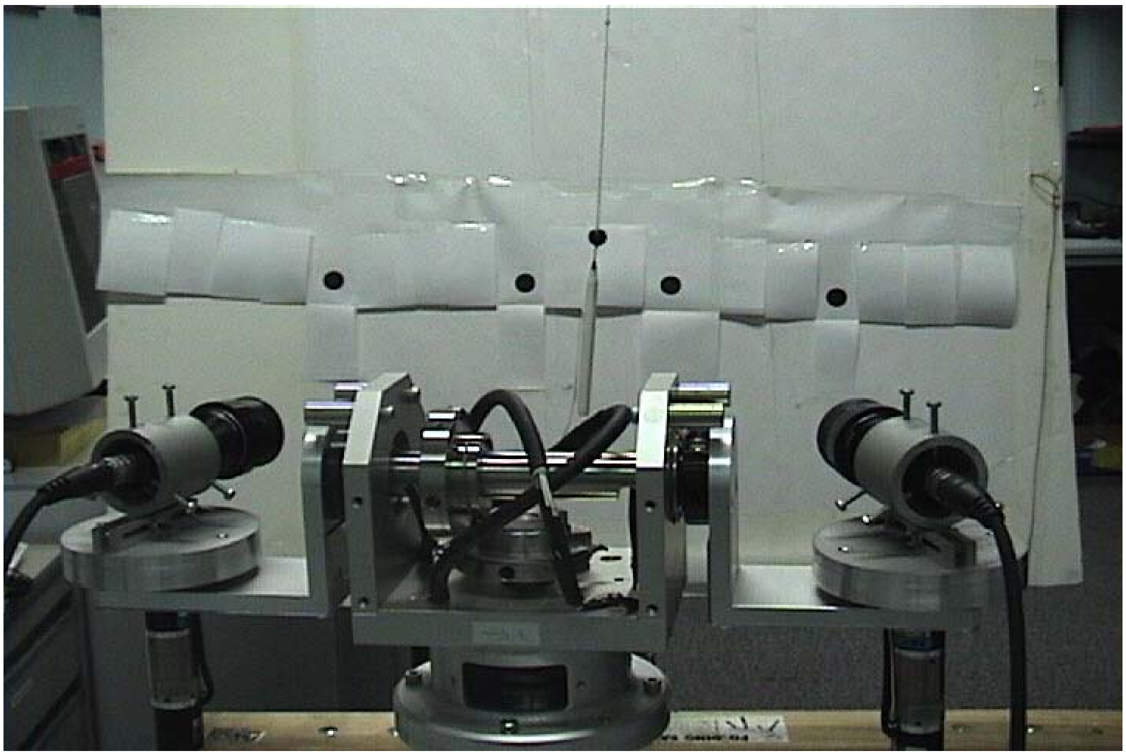


Figure 7.9: HyDrA and training apparatus to generate both smooth and step changes in image position

target, rather than switching randomly between the visible targets. However, if the current target was lost, the tracking process searched again over the entire view for the best matching target and selected a new current target.

Figure 7.10 illustrates the process. The sequence is also included as a movie file on the CD-ROM that accompanies this thesis. The current target was the fixed black circle on the right-hand side. As the lead weight swung to the right, the white cylinder obscured the current target. The lead weight was selected as the new target; another static target could also have been selected. If the controller lost the moving target a static target would again be selected.

To allow training with only static targets the lead weight could be covered so that it did not match the target template. The white cylinder then caused switching between the static targets. The training apparatus provided only stereotypical movements. Exposure to a broad range of movements required supplementing the range of experiences by hand.

Almost any experience generation regime was sufficient for the controller to learn to vaguely follow the target, but the nature of the learning experiences made an enormous difference to the controller's capabilities. Strong variations and biases in performance were apparent. In particular, controllers with fast step responses lagged while following the swinging target, and controllers that pursued moving targets accurately at high speed often had slow step responses to static targets.

The following sections analyse the phenomenon through the evaluation of the performance of two controllers. The first controller was trained with static targets, the second with swinging targets. Both controllers were then evaluated with static and swinging targets.

7.5.3 Training with Static Targets—Evaluating with Static Targets

The controller learnt to reliably fixate to static targets in both directions within a few minutes. An image sequence showing fixation to a newly selected target is shown in Figure 7.11. The sequence is also included on the CD-ROM. A graph of the same response is shown in Figure 7.12.

The *joint position* in degrees was read directly from the encoder via a motion controller card. The *joint action* is the un-filtered output from the learned controller; it is the desired velocity in degrees per second of the joint for the next time step. The *target position* and *target velocity* are based on image data in pixels. In the interest of clarity, these were approximately converted to the same degrees and degrees per second units

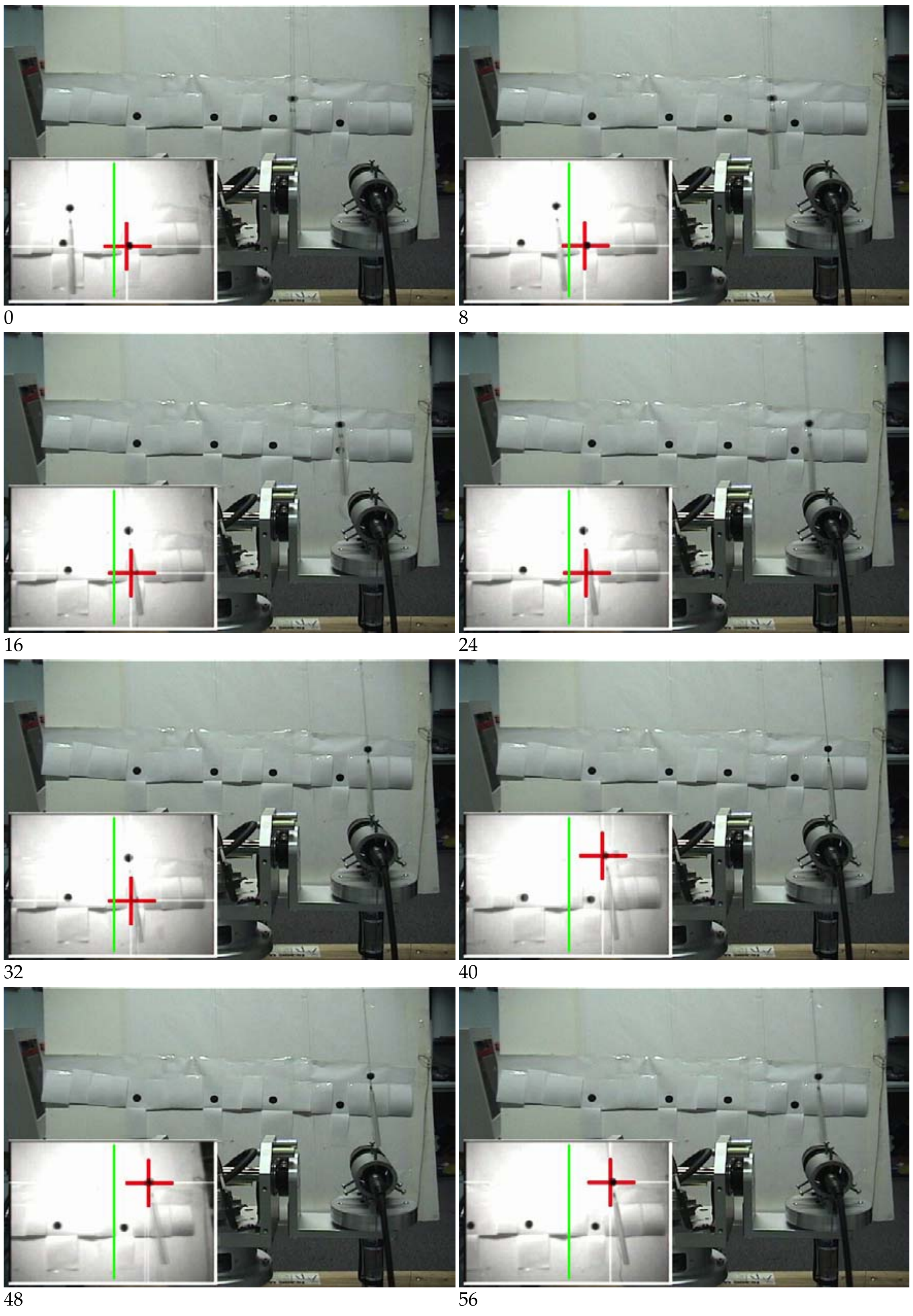


Figure 7.10: Transition between a static and a swinging target. The robot's view is inset. The numbers below each image indicate the number of frames at 30Hz; therefore images are around 0.26 seconds apart.

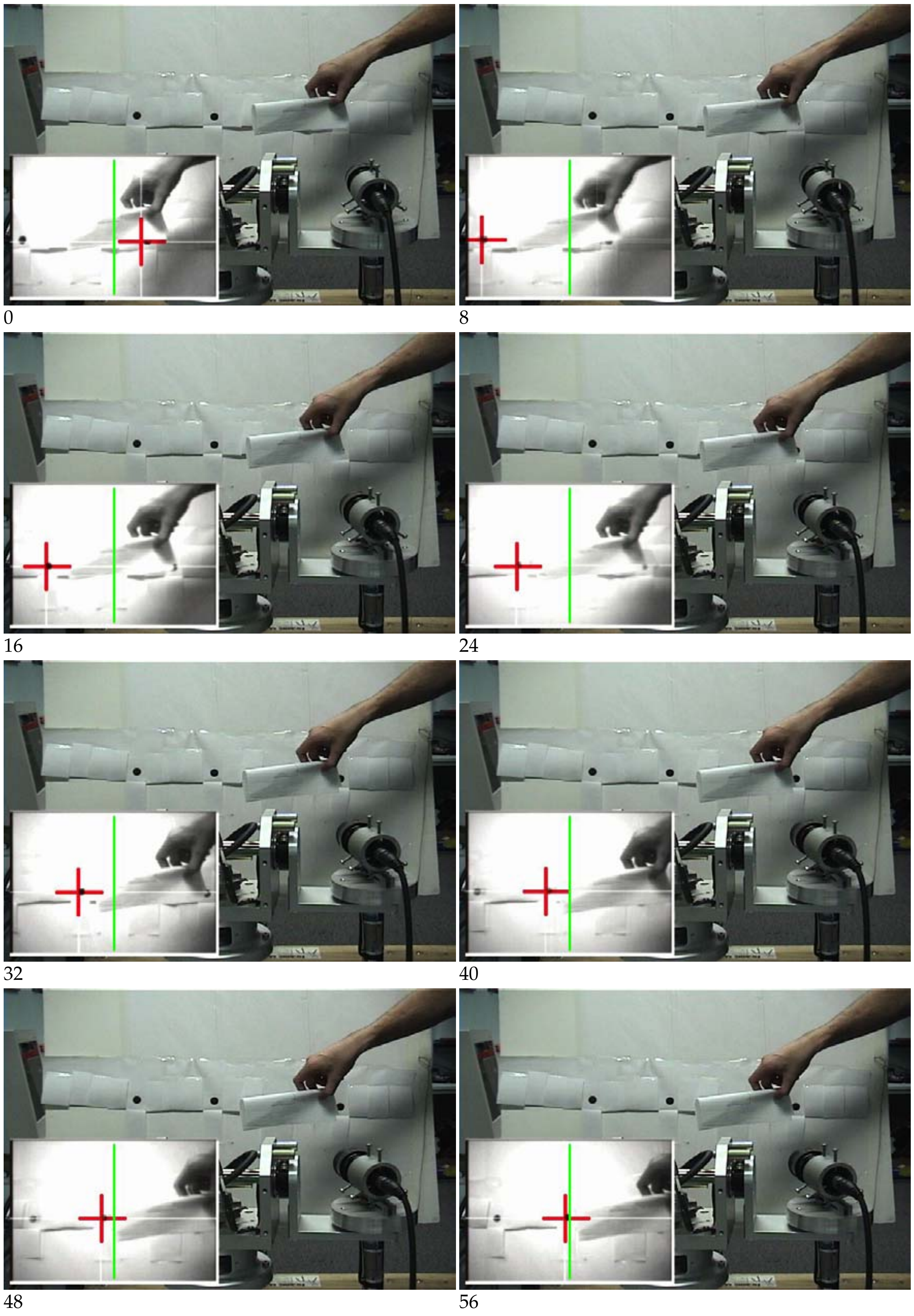


Figure 7.11: Static-trained controller pursuing a static target. The robot's view is inset. The numbers below each image indicate the number of frames at 30Hz; therefore images are around 0.26 seconds apart.

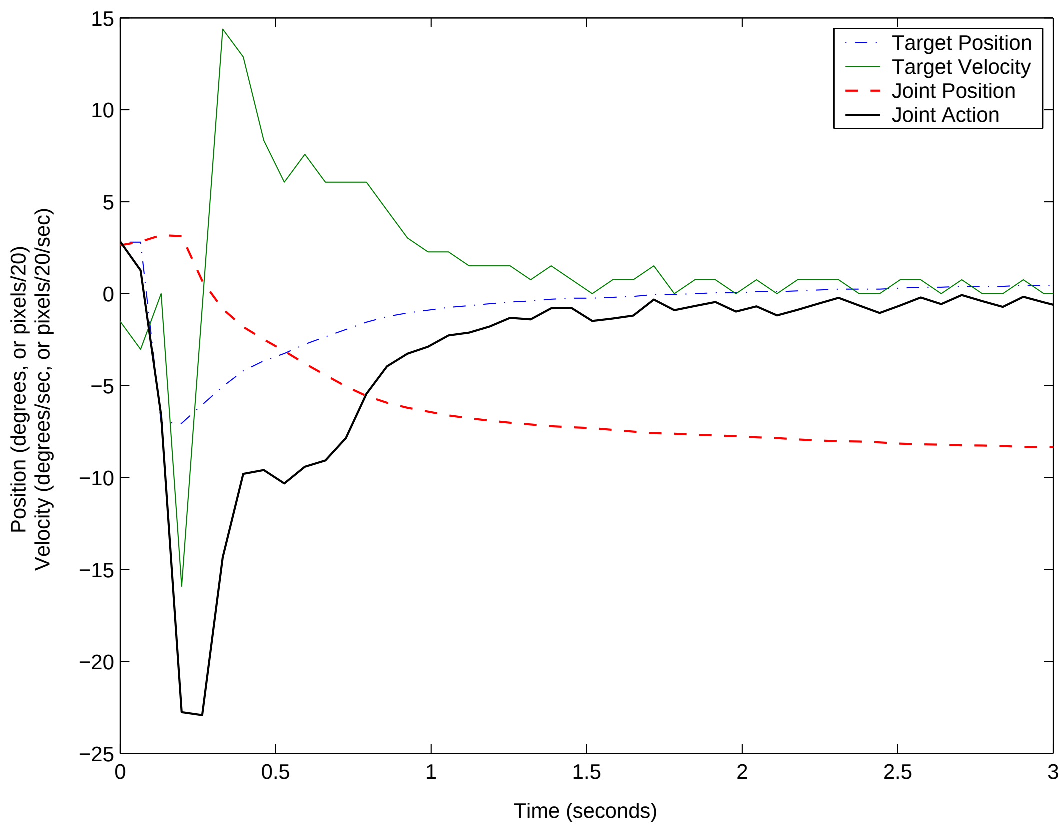


Figure 7.12: Static-trained controller pursuing a static target

as the joint position and joint velocity by dividing by 20. The scaling is based on physical measurements and the recorded data. The controller does not have access to the information; it is only used to present the graphs.

The figures show the controller's response in one direction only. The response was not exactly symmetrical: often the response in one direction was slower. This is a consequence of not including any symmetry constraints in the controller design. Chapter 4 suggested that the flexibility of a global approximator could allow the WFNN algorithm to perform high-level generalisations. The results of these experiments indicate that the algorithm did not generalise regarding symmetry in this case: it may have learnt to make step changes to the left and right independently.

Smooth joint movements

Smoothness of joint motion was varied by adjusting the weighting of the *smooth joint motion* and *pixels/second velocity of target* coarseness penalties in the reward function (see Figure 7.6). A heuristic method of adjusting these weightings is to start with a low penalty and determine if learning occurs. If learning occurs but the behaviour is coarse, increase the penalty. The converse method, starting with a high coarseness penalty, works poorly since the controller might move the joints slightly or not at all. In this situation it is difficult to tell whether the lack of movement was due to the high coarseness penalty, or complete failure of the learning system. Starting with a low coarseness penalty is no more dangerous for the hardware than a high coarseness penalty since when learning begins behaviour is random, independent of the reward function. The hardware must be protected by lower level filtering and vetoing mechanisms, as discussed in Chapter 5.

With appropriate coarseness penalties, the commanded action changed smoothly in response to the step change in target position. It is unlike the behaviour of a simple PID family controller and shows that the learned controller took the cost of acceleration into account. The learned controller accelerated more quickly than it decelerated, unlike HyDrA's existing trapezoidal profile motion (TPM) controller (Sutherland et al., 2000a).

The TPM controller accelerated with fixed acceleration to a maximum ceiling velocity, coasted at that velocity, then decelerated at a fixed rate equal to the acceleration rate. The learned controller's strategy of accelerating more quickly than it decelerates seems practical since final positioning accuracy is more dependent on the deceleration phase than the acceleration phase. Large saccadic motions in humans also accelerate more quickly than they decelerate (Becker, 1991). Eye velocity quickly reaches a saturation limit, reduces a little, then drops to zero slowly. The pattern creates a bump in the velocity profile between reducing a little and dropping to zero slowly. The learned controller also reproduced the bump in the velocity profile. Further comparison between the learned controller's step response and human eye saccadic motions is probably unwarranted since the learned controller acted in a closed-loop with respect to the image, whereas the image is not visible during human eye saccadic motions.

Saccadic joint movements

The active head does begin to perform open-loop, or saccadic, movements if the coarseness penalties are low. As the penalties were reduced the speed increased; consequently, the blurring of the target reduced the reliability of the target tracking and caused tracking failures. Ultimately, however, the learning system in its current form cannot succeed in performing saccadic movements because of the state and reward representation used. When target tracking failed learning ceased. When target tracking resumed (at the end of the saccade) learning resumed but there was no mechanism to pass rewards back to the actions that generated the saccade.

If the state representation were enhanced to allow saccadic motions representation of the position and velocity of the target when tracking fails would become a problem. Learning systems that use approximators make it difficult to simply remove invalid components. One approach is to include the tracking confidence in the state vector. Disadvantages of the approach are the increased number of state variables needed and that possibly misleading state information is included in the representation when the confidence is low. Another problem is choosing a reasonable reward function. Modifications to the reward function must also take into account loss of tracking due to the target being covered, moved, then revealed. It is difficult to represent the distinction between a saccade, in which tracking fails due to fast joint movement, and the target simply being covered, moved, then revealed.

7.5.4 Training with Static Targets—Evaluating with Swinging Targets

Figures 7.13, 7.14, and 7.15 show the performance of the controller trained with static targets when pursuing a swinging target. The sequence is also included on the CD-ROM. Learning was manually disabled during these experiments so that the controller must perform the task based only on its experience with static targets.

The interaction between gaze control and motion blur was readily observable. If the active head was still and the target swings past quickly the template matching mechanism could not consistently locate the blurred target. However, once the active head was following the target it was tracked easily at high velocities without blurring. The background was blurred instead.

When attempting to follow smoothly moving targets there was some lag. Lag is inevitable due to sensing and actuation delays unless the controller can predict the movement of the target. The lag is especially noticeable in Figure 7.13 when the target's direction changes, but Figure 7.14 shows that the positioning error is highest at the bottom of the swinging motion, when the target's velocity is highest. Ideally the target position and velocity should be maintained at zero. Using our controller there was some variance in the target position and velocity but the bias was small.

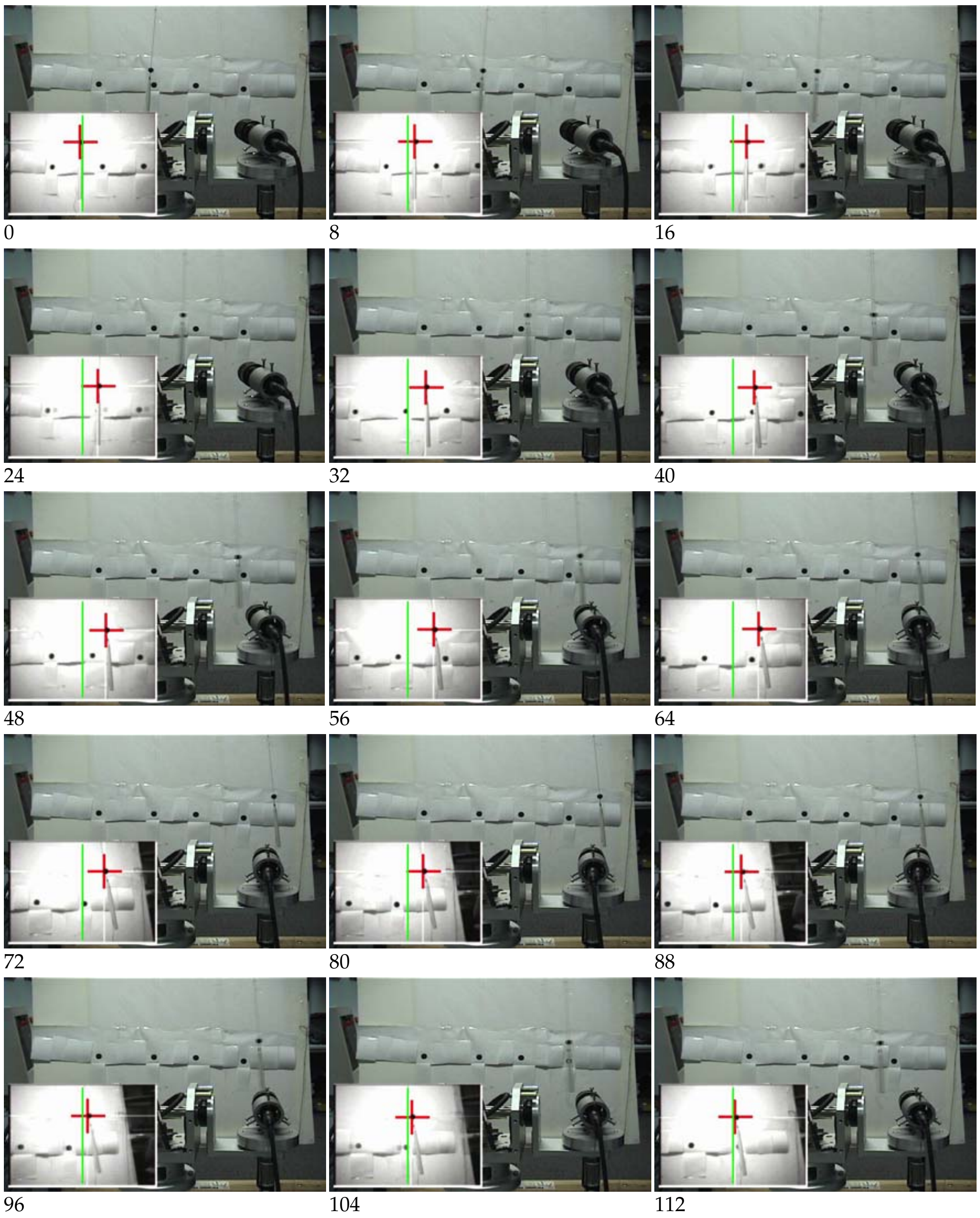


Figure 7.13: Static-trained controller following a swinging target. The robot's view is inset. The numbers below each image indicate the number of frames at 30Hz; therefore images are around 0.26 seconds apart.

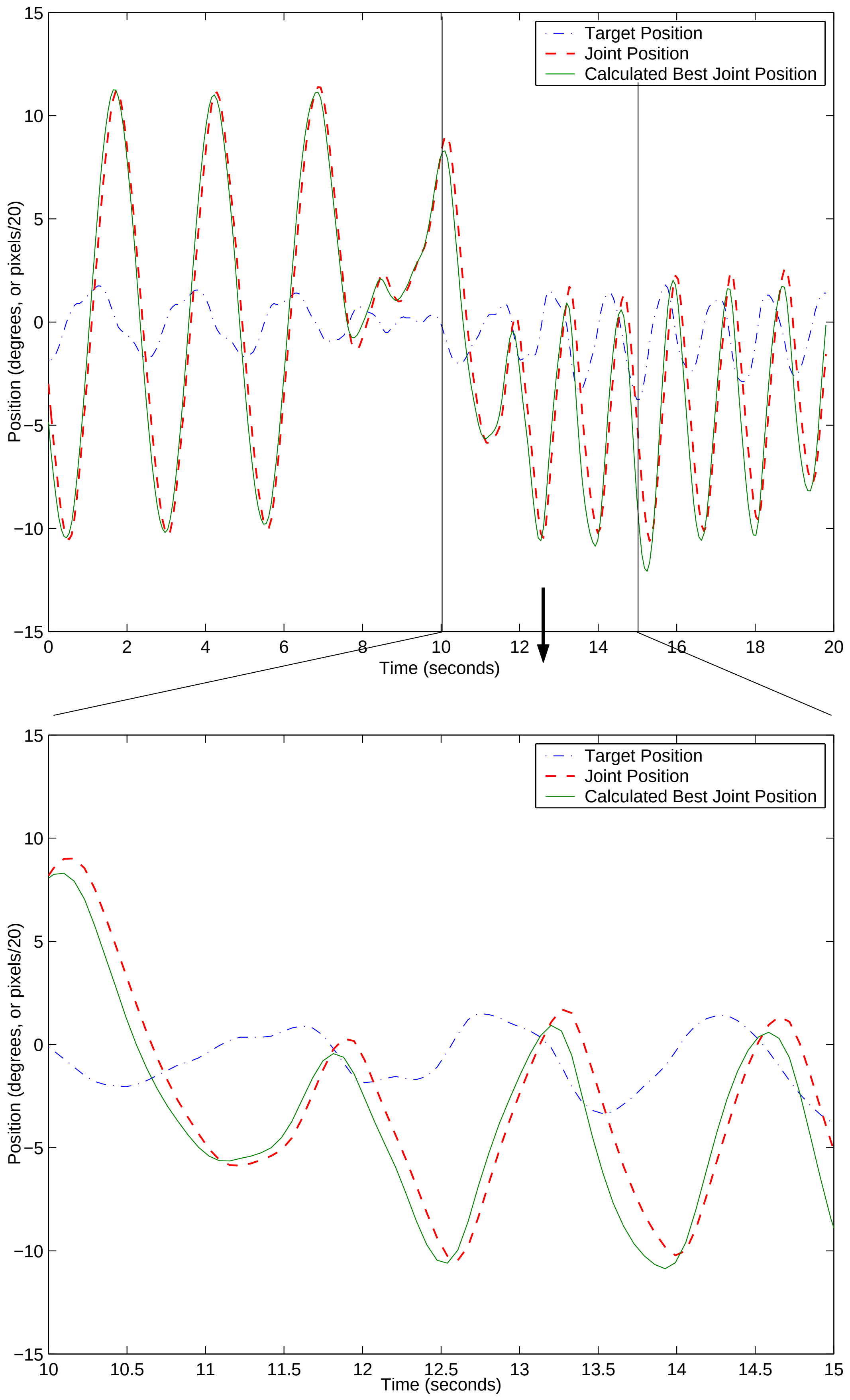


Figure 7.14: Position data from the static-trained controller following a swinging target

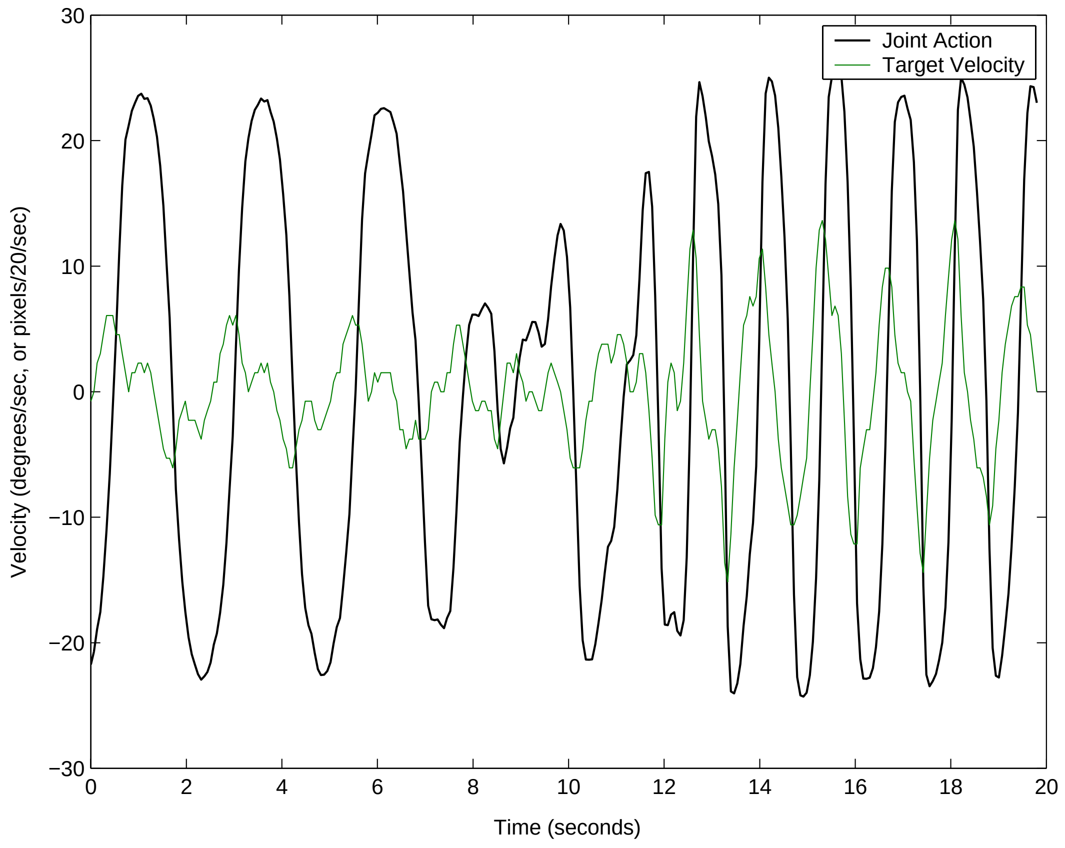


Figure 7.15: Velocity data from the static-trained controller following a swinging target

7.5.5 Training with Swinging Targets—Evaluating with Swinging Targets

When trained only with swinging targets the controller learnt to smoothly follow the target with excellent performance within a few minutes. An image sequence showing pursuit of a swinging target is shown in Figure 7.16. The sequence is also included on the CD-ROM. The image sequence and the graphs in Figures 7.17 and 7.18 show that the swing-trained controller eliminated the pursuit lag that was seen with the static-trained controller. The swing-trained controller had small target velocity bias, target velocity variance, and target position variance. However, there was some bias in the target position; the bias was largest when the target was swung at a higher than normal velocity in a different position. At all times the target velocity bias and variance remained small.

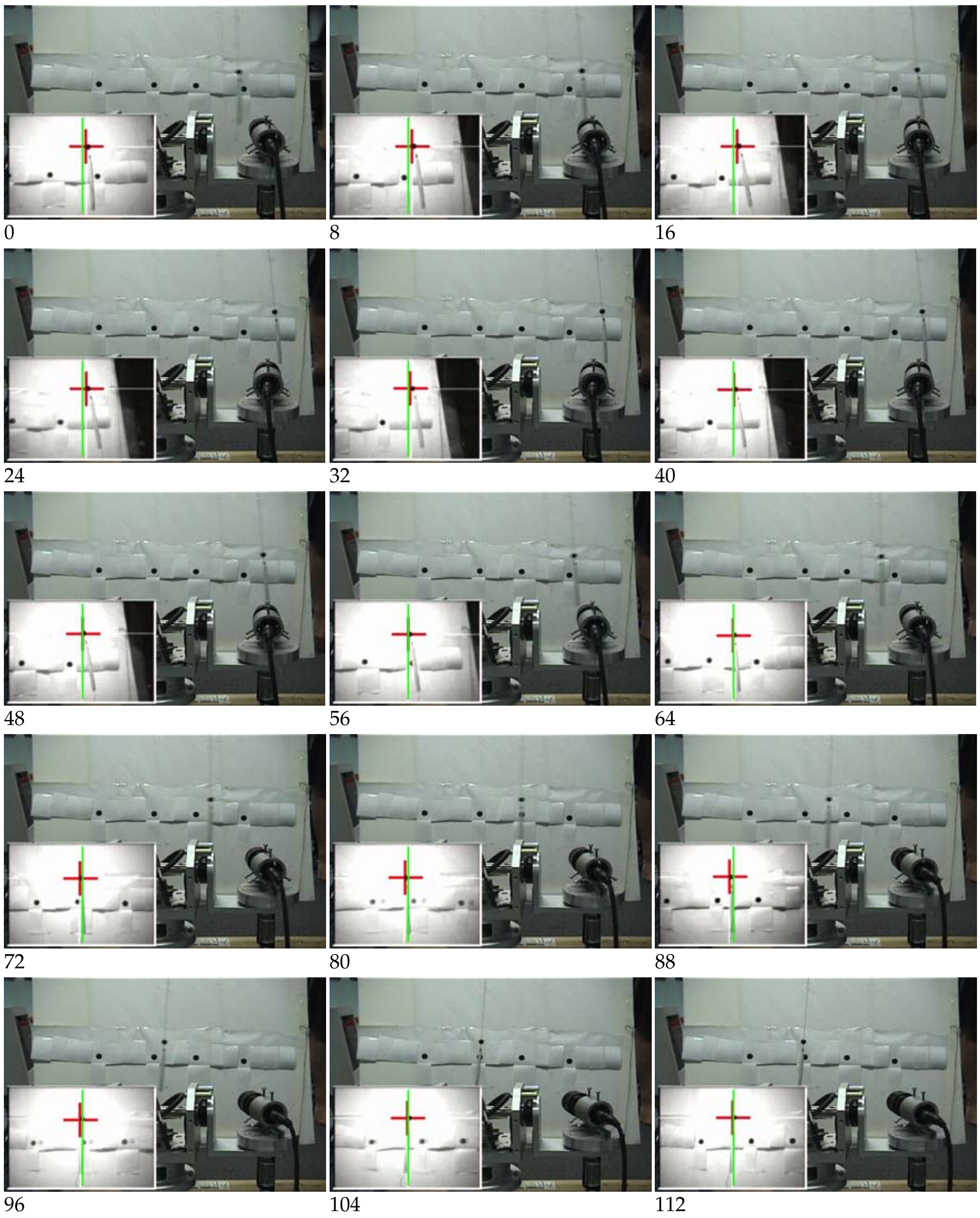


Figure 7.16: Swing-trained controller following a swinging target. The robot's view is inset. The numbers below each image indicate the number of frames at 30Hz; therefore images are around 0.26 seconds apart.

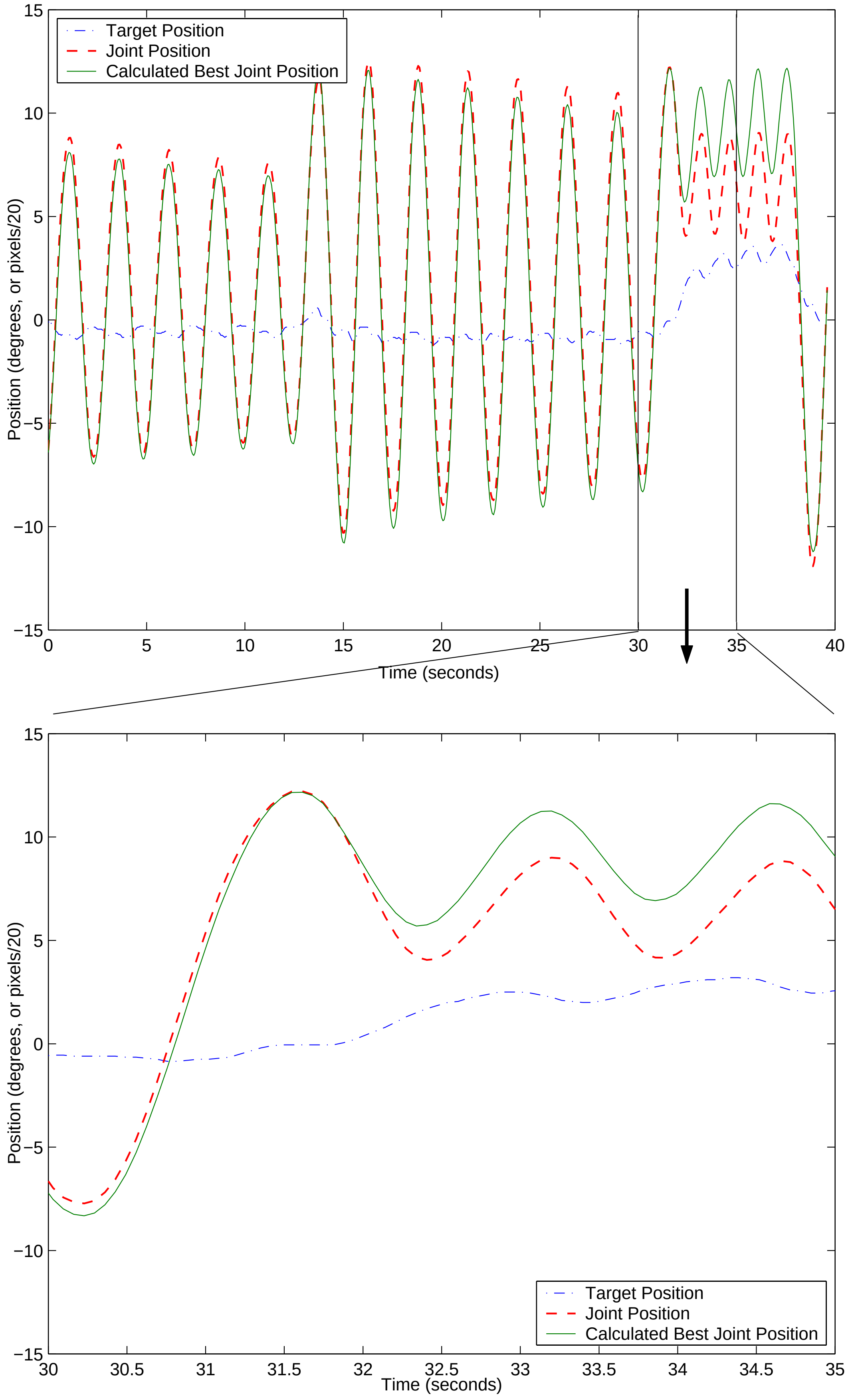


Figure 7.17: Position data from the swing-trained controller following a swinging target

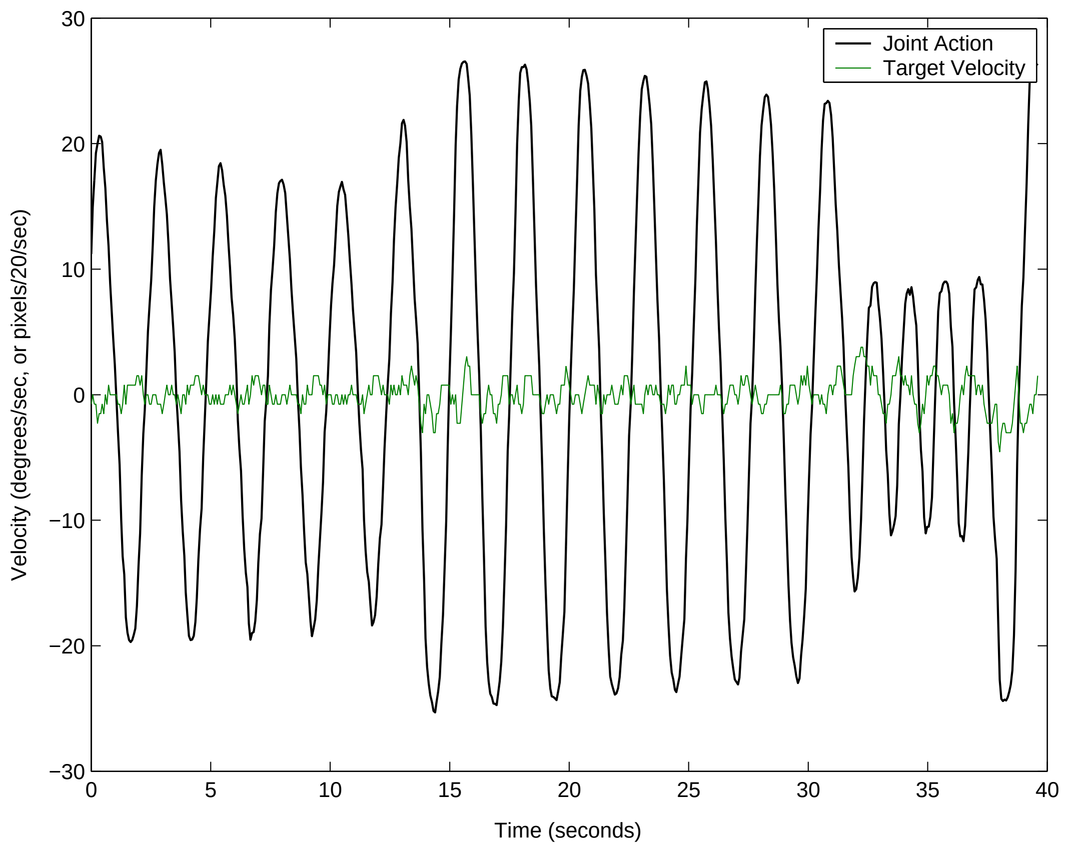


Figure 7.18: Velocity data from the swing-trained controller following a swinging target

Elimination of delays

The *static-trained* controller described earlier was affected by delays when pursuing a swinging target. These delays are inevitably introduced by vision processing and actuation. Other controllers for active heads have included explicit mechanisms for coping with delays, either by modelling their effect and attempting to compensate (Shibata and Schaal, 2001a; Brown, 1990; Murray et al., 1992, 1995), or by learning to predict the behaviour of the target (Shibata and Schaal, 2001b). The *swing-trained* controller was able to eliminate pursuit delay for swinging targets, even though the learning system, state and action representation, and reward signal were the same as for the static-trained controller. The only difference between the two controllers was the dynamics of the environment that the controllers were exposed to. It seems that the swing-trained controller learnt to predict the motion of the swinging target. By predicting the motion the swing-trained controller was able to eliminate delays and gain higher rewards—assuming that the target continues to swing.

Controller analysis

To verify the theory, we compared the actions of the control system to the dynamics of a pendulum model. The pendulum model was developed based on measurement of the weight of the target hanging on the string, the length of the string, and the gravitational constant. Incorporating the approximate relationship between the robot's joint angle and the angle of the string gives the required joint acceleration to match target acceleration due to gravity[1] :

$$\ddot{\beta} = \frac{\beta g}{l}\sqrt{1 - \left(\frac{\beta d}{l}\right)^2} \tag{7.1}$$

Comparison of the model to the behaviour of the controller does not necessarily require analysis of experimental data gathered using the robot. It is not possible to investigate the controller's behaviour over *multiple* time steps without a dynamic model of the environment, but the controller's behaviour over a *single* time step can be extracted directly from the controller itself by providing an appropriate state input. In this case the state vector has joint velocity, target position, and target velocity of zero. The joint position is the independent variable; the dependent variable is the joint acceleration, which was calculated from the initial joint velocity and output joint velocity. Figure 7.19 shows joint acceleration for various joint positions. It compares the output of the swing-trained and static-trained controllers with the pendulum model and a static object model ($\ddot{\beta} = 0$ radians/s^2). The output from the swing-trained controller resembles the pendulum model, while the output from the static-trained controller is similar to the static model. This evidence supports the theory that the controllers are predicting the behaviour of the target.

7.5.6 Training with Swinging Targets—Evaluating with Static Targets

Figure 7.20 is an image sequence showing the swing-trained controller's behaviour when exposed to a static target. The sequence is also included on the CD-ROM. Learning was again disabled so that the controller must perform the task based only its experience with swinging targets. The target was static between frames 0 and 24 and for around 10 seconds before frame 0. During that time the active head was nearly

[1]Equation (7.1) is an approximation, and based on rough measurements of the pendulum string length and the distance from the camera to the target at rest. The variables are $\ddot{\beta}$ radians/s^2, the joint acceleration to match target acceleration; and β radians, the joint angle to match pendulum angle. The constants are $g = 9.81$m/s^2, the acceleration due to gravity; $l = 1.6$m, the length of the pendulum string; and $d = 0.64$m, the distance from the camera to target at rest.

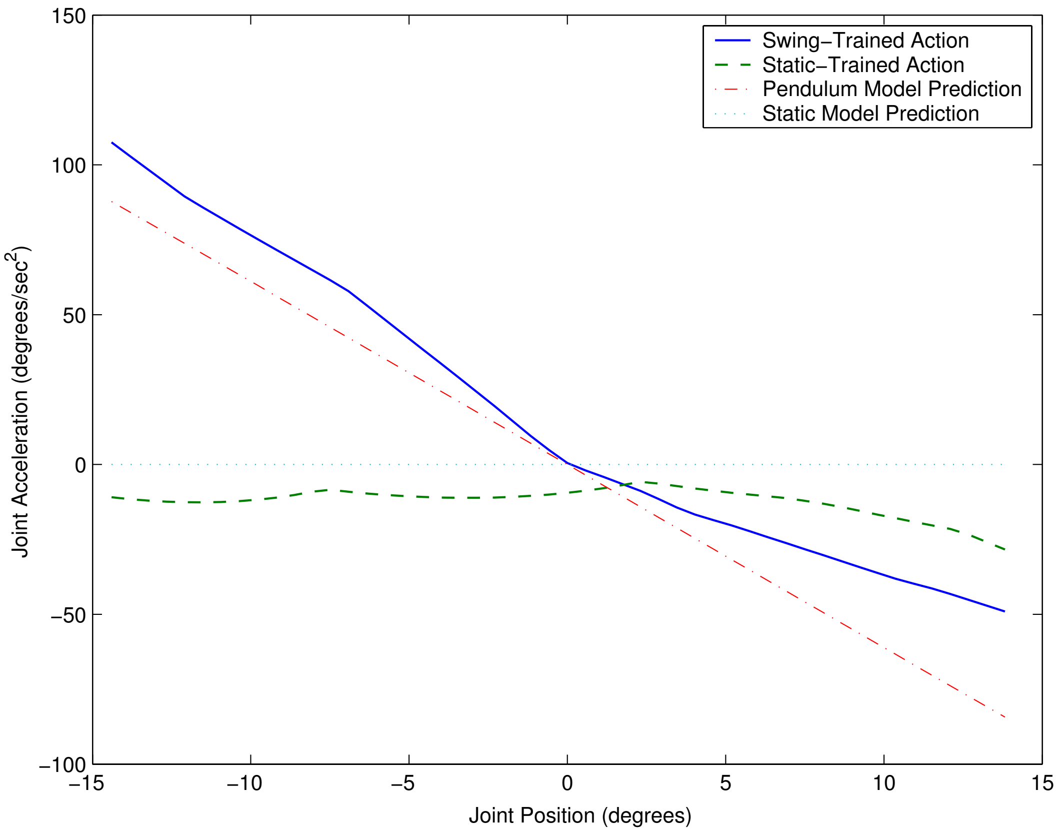


Figure 7.19: Actual output and model-predicted joint acceleration for both the static-trained and swing-trained controllers. Joint velocity, target position, and target velocity are zero.

static. The swing-trained controller failed to pursue the static target. After frame 24 the swinging target was released and the controller successfully pursued the target. The behaviour was repeatable across training sessions.

The controller rotated the camera towards the right during the first swing, matching the target velocity rather than immediately eliminating the target position bias. The target position bias was eliminated when the target was about to swing back. The graph in Figure 7.21 shows the same situation. The graph shows that the joint was wiggling slightly while the target was static. The large downward spike in target velocity was due to the release of the target. The lower (zoomed) portion of the graph shows that the controller was not responding to the change in target position as the target was released; it was responding to the velocity of the target. The upward spike in target velocity was due to the an overshoot in the joint velocity as the controller matched the velocity of the target.

Another unexpected behaviour sometimes occurred if a pursued swinging target was seized near the extreme of its swing. The controller did not fixate the camera con-

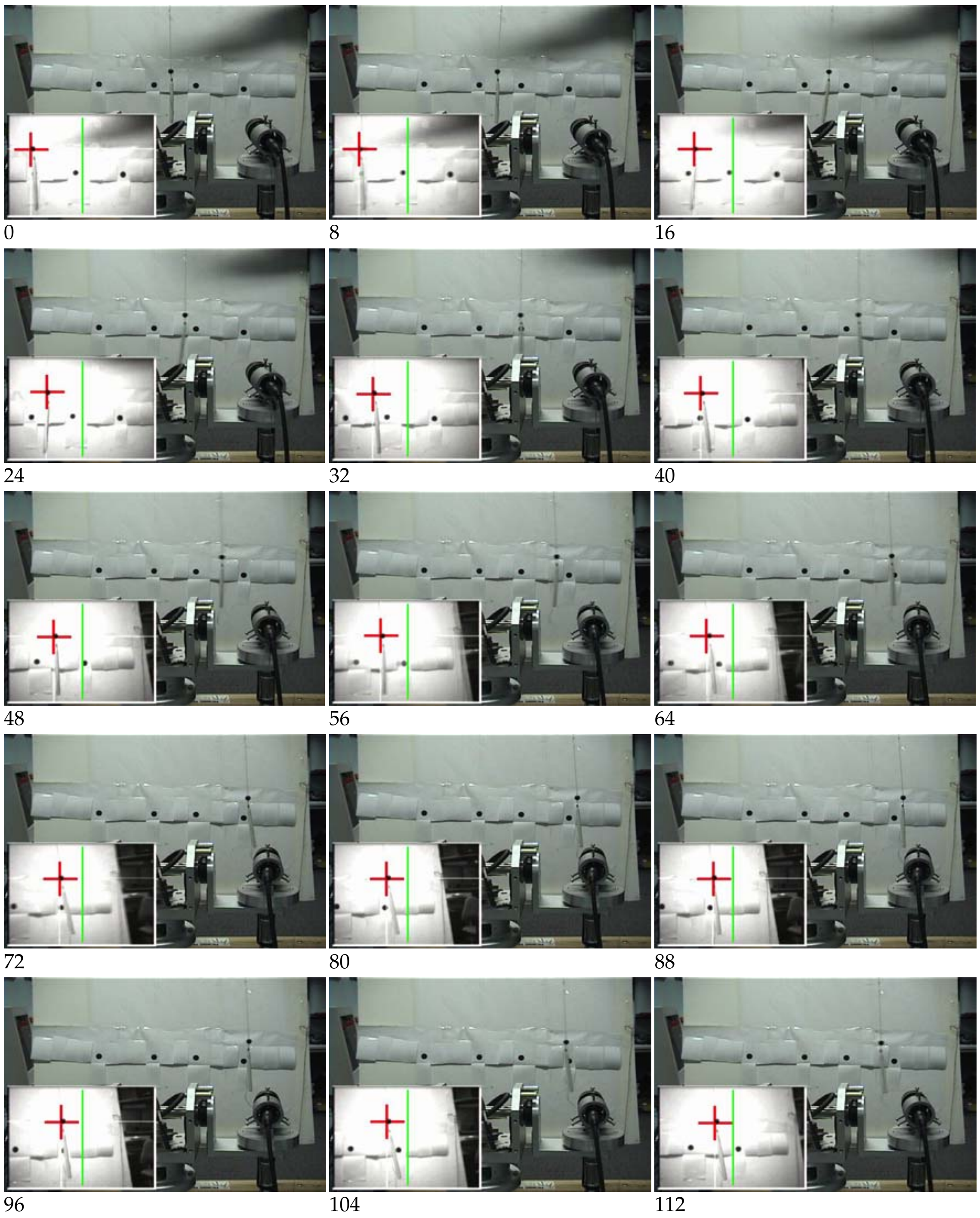


Figure 7.20: Swing-trained controller failing to pursue a stationary target. The robot's view is inset. The numbers below each image indicate the number of frames at 30Hz; therefore images are around 0.26 seconds apart.

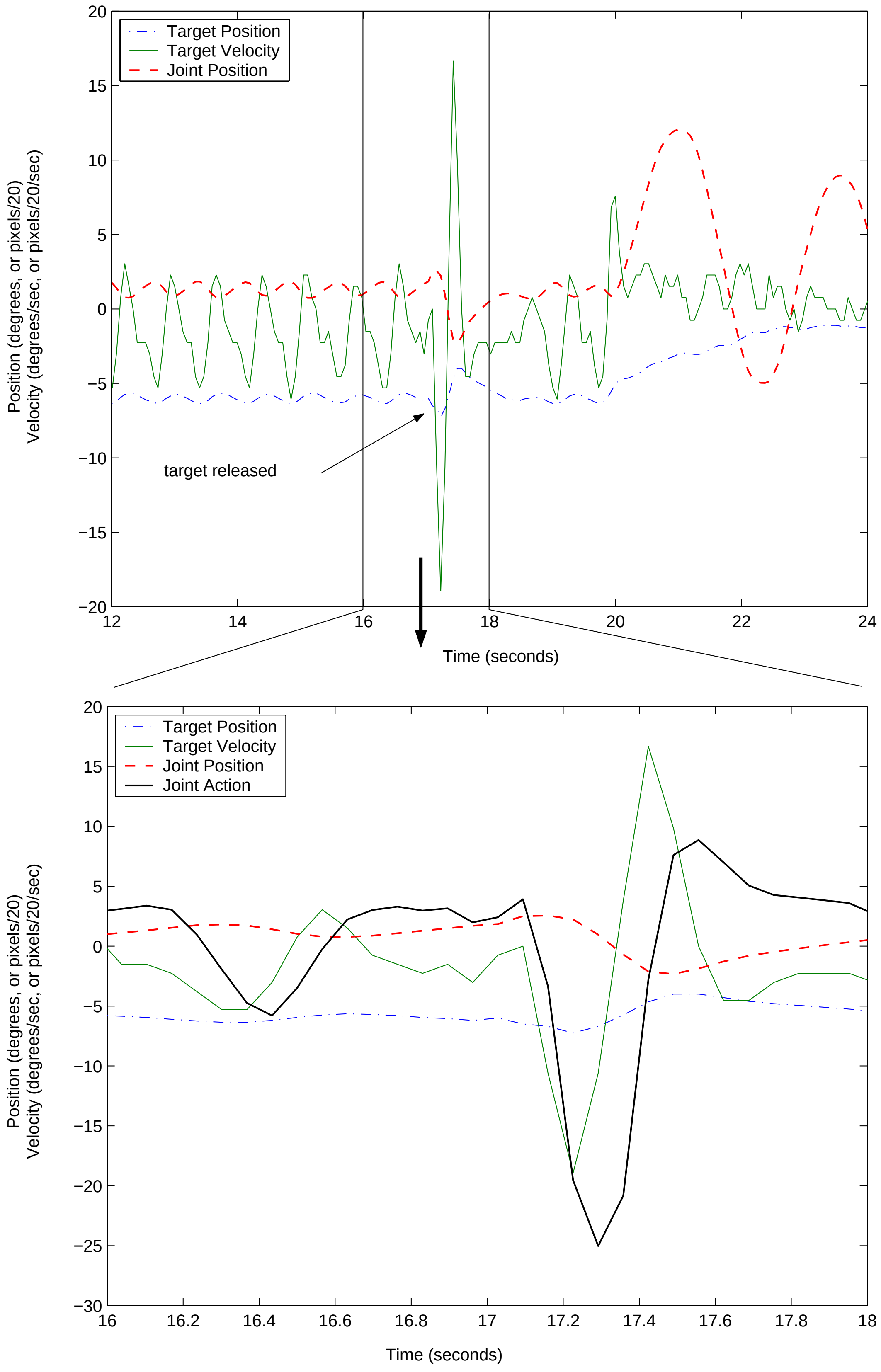


Figure 7.21: Swing-trained controller failing to pursue a stationary target

sistently on the now still target: it moved the joint back a few degrees towards the central position, stopped, moved to fixate on the target again, stopped, then repeated the process. Sometimes the controller made several of the ticking motions then stopped, fixated on the target. On other occasions the ticking motion was repeated, interspersed by pauses of variable length, until the target was released.

Controller analysis

The behaviour of the swing-trained controller when exposed to static targets is strong evidence for the theory that the controller is predicting target behaviour. When faced with a static target the swing-trained controller seemed to act as if predicting swinging behaviour—waiting for the target to swing towards the middle, and moving in expectation that the target will swing towards the middle.

As in the previous section, we validated the theory further by synthesising appropriate state data and analysing the controller's output. In this case the state vector has joint velocity, joint position, and target velocity set to zero. The target position is the independent variable. As before, the dependent variable is the joint acceleration. The configuration imitates the appearance of a new, static target. Figure 7.22 shows the output from the static-trained and swing-trained controllers. The static-trained controller moves in the direction of the target. The swing-trained controller moves inconsistently: if the target is to the right it moves towards the target, but if the target is to the left it moves away! The result shows that the swing-trained controller was specialised towards pursuing swinging targets and was not capable of fixating to static targets.

The single degree of freedom experiments investigated the response of the learning system to different experience and evaluation regimes. The following section addresses the problems of multiple degree of freedom control: scaling the learning system to more complex problems, and the redundancy introduced by the pan and verge-axis joints.

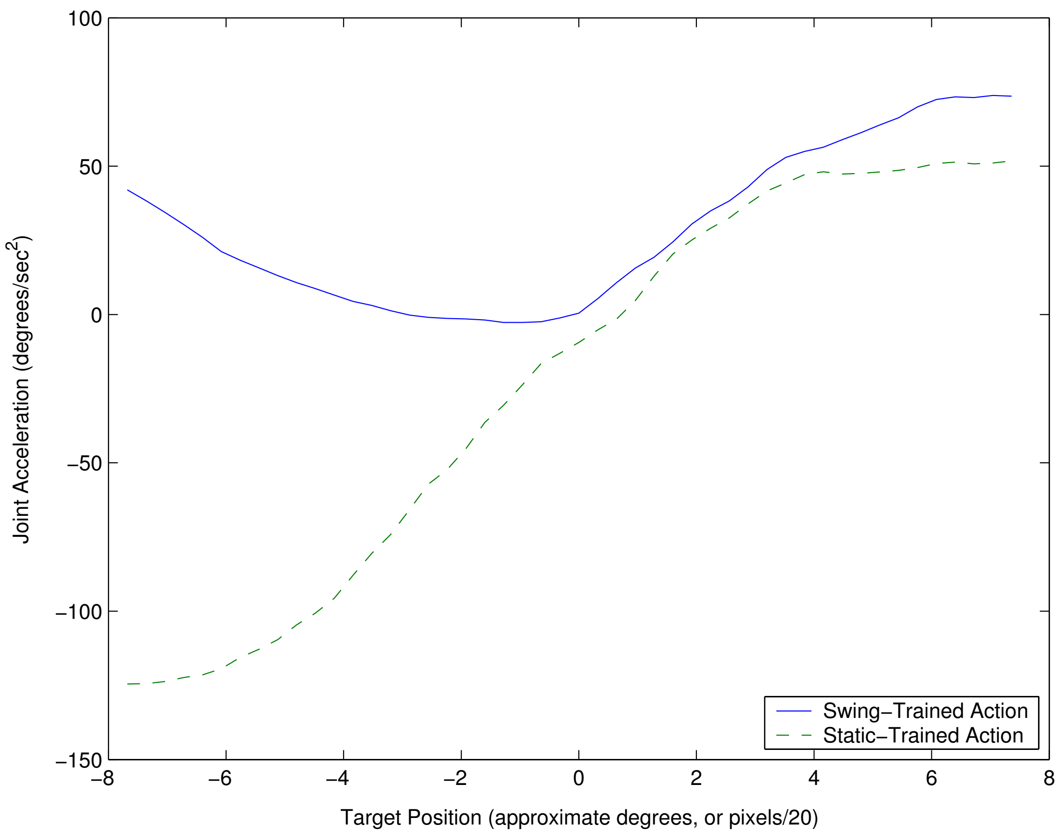


Figure 7.22: Joint acceleration for both the static-trained and swing-trained controllers when exposed to a static change in target position. Joint position, joint velocity, and target velocity are zero.

7.6 Four Degrees of Freedom Visual Servoing

This section describes experiments performed with the 4 joints and 4 visual axes configuration, as in Figures 7.5 and 7.8(b). No change to the controller was required, only a change to the list of included joints and image axes.

Figure 7.23 shows an example visual servoing sequence using all four joints and all four camera axes. An example sequence is also included on the CD-ROM. Reasonable performance was achieved after at least 20 minutes of good quality interaction with the target. The learned controller performed fixation movements to static targets, and pursuit of moving targets, using all four joints. The controller performed tilting movements as the target moved up and down, vergence movements as the target moved towards or away, and versional movements as the target moved from side to side.

The string suspending the target was replaced with elastic to allow movement in three dimensions. Unfortunately, the elastic introduced some risk to bystanders given that the target was made of lead. A further disadvantage was that attempting to move

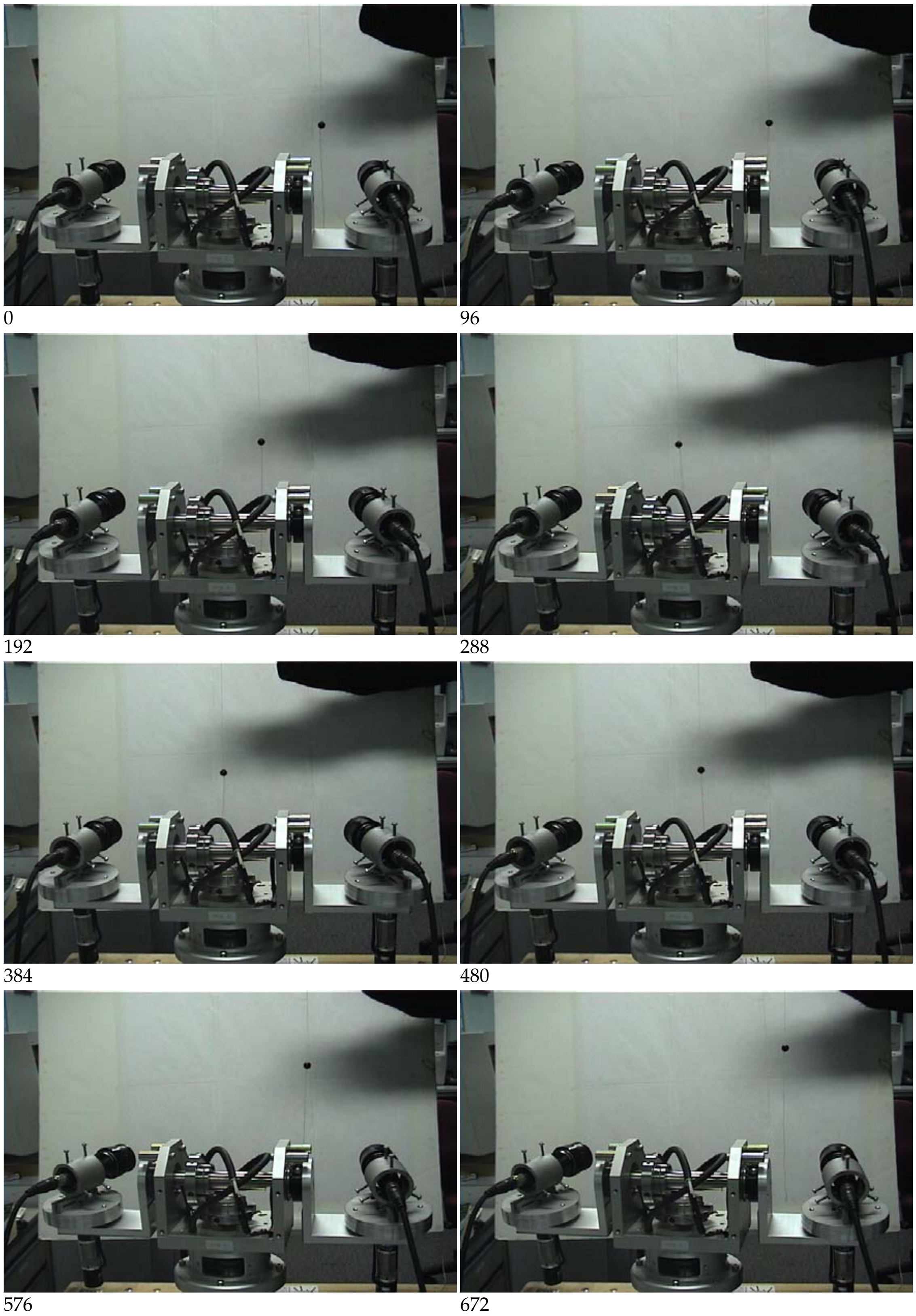


Figure 7.23: Visual servoing with four degrees of freedom and binocular vision. The target was a lead weight suspended on a piece of elastic. The numbers below each image indicate the number of frames at 30Hz; therefore images are around 3.2 seconds apart.

an elastically suspended target manually as a static target often resulted in uncontrollable high-frequency oscillation of the target, generating spurious data. A target mounted on a stick was used instead when a static target was required.

The training apparatus described earlier was unsuitable for binocular experiments as it is likely that different targets will be chosen from each camera's view. A more suitable apparatus would ensure that the same target is selected by dynamically marking the current target, for example by lighting an LED, or by making the targets different in appearance and selecting the target in software.

Binocular vision introduced a further difficulty. As described earlier, learning ceased if the target passes out of view of either or both of the cameras. In the monocular configuration this was only a slight inconvenience as the target can easily be moved back into the single camera view. However, in the binocular case there is no built-in constraint to prevent the verge-axis joints rotating so that the cameras diverge to point away from each other. Camera divergence occurred often in the early stages of the learning process and caused significant inconvenience. Later, the divergence ceased, even if one or both cameras do not have the target in view, or if the cameras have erroneously acquired different targets and divergence would be expected. Thus, the learning system discovered that the cameras should never be pointing apart.

Is not possible for the controller to fixate perfectly due to mechanical imperfections. For example, if the left and right cameras are posed at different tilt angles using their adjustable mountings then the target cannot be exactly in the centre of view for both cameras. In this situation the controller learnt to position the target in a compromise position: slightly high in one camera's view, and slightly low in the other camera's view.

7.6.1 Coping with Redundancy

The learning system sometimes had trouble exploiting the redundancy between the pan and verge-axis joints. For example, the pan joint could be held stationary while the verge-axis joints do most of the work, or one verge-axis joint could be held stationary while the pan joint and the other verge-axis joint work. Examples are shown in Figure 7.24. Occasionally one joint moved in the opposite direction to that expected, while other joints compensated.

These unexpected behaviours usually disappeared with further learning but sometimes biases remained, for example holding the pan joint offset from the central position. This is understandable since the reward function did not express any preference for par-

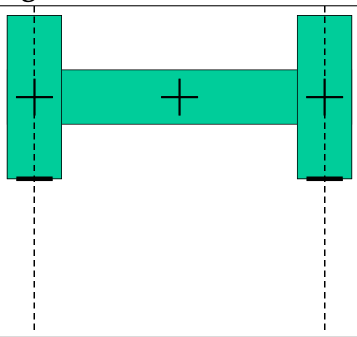
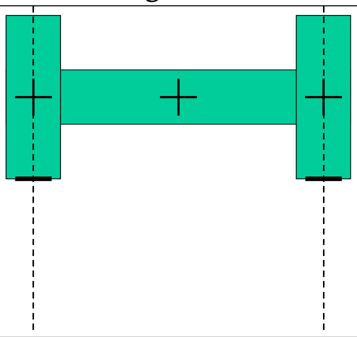
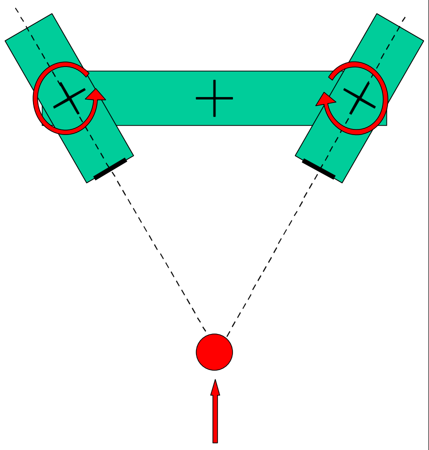
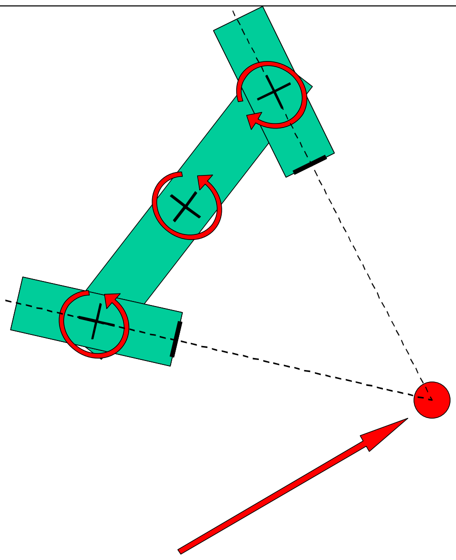
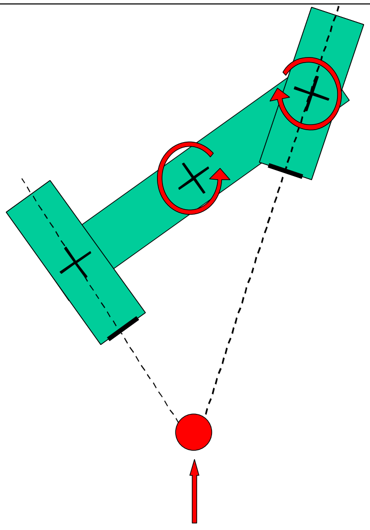
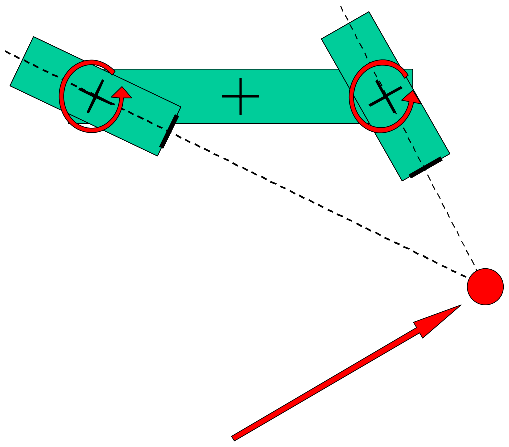
	Verge	**Version and Verge**
Initial position		
Expected Motion		
Example Learned Motion		

Figure 7.24: Examples of expected and learned motions. Arrows show the directions of target movement and joint rotation.

ticular positions of the joints—if the target is in the centre the field of view any joint pose is acceptable. In some ways keeping one joint stationary is a simpler control approach because it reduces the degrees of freedom. The main problem with the strategy is that the range of moment is restricted. If the full range of movement is not required during learning there is no information to lead the controller to avoid learning restricted movement controllers.

The redundancy can be effectively eliminated by modifying the reward function to specify a different task for the pan joint. For example, in Berthouze et al.'s (1996) controller the task for the pan joint was to minimise the difference between the horizontal position of the target in the left and right camera images. The drawback is that it limits the range of possible controllers and makes assumptions about the active head's mechanical configuration.

7.6.2 Coping with Instability

During the learning process the learning system sometimes lost stability and diverged. Divergence occasionally occurred even in the single degree of freedom case; however, it became a persistent problem in the four degree of freedom configuration. The divergence appears to be due to the rising $\mathcal{Q}$ effect described in Chapter 4. Without the use of advantage learning divergence always occurred within 10 seconds of starting the experiment; with advantage learning divergence rarely occurred within 5 minutes of starting.

Since no method for preventing or curing divergence was available, a coping strategy of detecting divergence then trying to make the best possible use of gathered data was a reasonable option. In our experiments, divergence was detected through monitoring the expected value of chosen actions. Since the maximum reward is zero the expected value of actions should not rise above zero. Values above zero can be expected occasionally during the learning process, but sustained sequences of high values always indicate a problem. If the value of chosen actions consistently rose above 0.5 the controller was judged to have diverged beyond redemption.

After divergence occurred the current weights of the neural network were useless—to make a fresh start they were reset to small random values. However, the buffer of experiences was still valid and was a valuable resource for learning. Using the *experience replay* approach described in Chapter 2 the learning system learnt from the data passively, while simultaneously improving its performance actively through interaction with the environment.

During the learning process around 20 divergences and reset cycles could occur before a stable configuration of weights was found. Once the buffer of stored experiences had grown sufficiently large, learning took place quickly, sometimes in less than 10 seconds. This shows the value of gathered experience, and is evidence that off-policy learning and the passive learning approach are worthwhile.

7.7 Discussion

Our aim was to learn a multiple degrees of freedom controller capable of smooth pursuit of moving objects as well as smooth movement towards a just recognised static target. Providing experience for learning was difficult, especially with many degrees of freedom, because the target was often lost and learning ceased. One cause of the problem is that the controller is learning from scratch, rather than building on an existing, semi-competent controller (as in Berthouze et al., 1996; Shibata and Schaal, 2001a).

Although the four degree of freedom controller demonstrated various competencies it usually failed to converge to a controller that was competent overall. There were always regions of the state space in which the controller performed poorly enough to lose the target. For example, a controller could fixate and pursue targets competently toward the left, but ignore targets to the right.

Even for the one degree of freedom problem, learning a controller that was competent at both pursuing moving targets and fixating on static targets was quite difficult. The controller always seemed to be competent at some parts of the task but not others. More consistent results were obtained by dividing the task between two controllers: the first specialising in step movement to a target (the static-trained controller), the second specialising in smooth pursuit (the swing-trained controller). The division of labour is similar to the human vision system's saccade and smooth pursuit systems. A major difference is that the static-trained controller operates in a closed-loop manner, whereas the human saccadic system is open-loop. Although motivated by the results of the first experiments and the biological precedent the task division was also a result of practicality constraints: it was labour-intensive to provide a mixture of step and smooth movements as training examples.

Table 7.1 summarises the performance of both one degree of freedom controllers. Although the static-trained and swing-trained controllers had the same initial conditions, the learned controllers differed because the controllers were exposed to different experiences. This resulted in qualitatively different behaviour. The difference was emphasised

Table 7.1: Summary of single degree of freedom results

trained/evaluated	**static**	**swing**
static	pursuit (Figures 7.11, 7.12, and 7.22)	pursuit with lag (Figures 7.13, 7.14, 7.15, and 7.19)
swing	no pursuit (Figures 7.20, 7.21, and 7.22)	pursuit without lag (Figures 7.16, 7.17, 7.18, and 7.19)

when the controllers were applied to tasks other than their speciality.

The results showed that the controllers learnt to predict the behaviour of the target. There are biological parallels in the human smooth pursuit system. The human smooth pursuit system is basically driven by target velocity (Rashbass, 1961), but predictive, adaptive mechanisms are used to increase efficiency. Jürgens et al. (1988) investigated to the predictive system and discusses several explanations for these mechanisms. After exposure to periodic target movements humans can pursue a target with zero lag, despite at least 100ms of delay in the human vision processing system. An internal model of target movement is formed to increase the accuracy of the smooth pursuit. If the target becomes invisible pursuit can continue for several seconds, following the periodic pattern. If the target then becomes visible pursuit can continue without lag. Jürgens et al. also suggested that the smooth pursuit system may use the path of targets travelling towards the edge of the field of view to estimate the return path of the target as it moves toward the centre of the field of view. The mechanism seems to be active even for randomly moving targets. The human vision system is able to adapt in real-time as the target's behaviour changes. Shibata and Schaal's (2001b) active head controller had an *explicit* module for learning target behaviour that was capable of adapting to patterns in target movement in only a few seconds. Although our active-head controller did not have this ability it did learn slowly.

The WFNN algorithm does not represent the target's behaviour or the dynamics of the active-head mechanism explicitly. They are represented *implicitly* through the stored action-values. The state representation shown in Figure 7.6 makes no distinction between state variables of the robot, such as joint velocity, and state variables from the environment, such as target position. There is no explicit connection between the actions and the related state variables. The learning algorithm is interacting with a meta-environment consisting of the robot and its environment, in which much of the relevant state information is unmeasured and not represented by the state vector. However, the $\mathcal{Q}$-learning algorithm assumes interaction with a Markov environment, that is, the probability distribution of the next-state is affected only by the execution of the action in the current, measured state.

Consequently, the swing-trained controller learnt to predict the acceleration of the swinging target based on the joint position (Figure 7.19). Inclusion of the joint position in the state information was necessary because the joint behaves differently at the extremes of its rotation range. Predicting target behaviour based on joint position is obviously flawed since the model is no longer applicable if the position of the pendulum's anchor changes (i.e. the position at rest changes). This illustrates a flaw in the model-free learning system paradigm: failing to separate controllable mechanisms from uncontrollable environment can lead to learning a controller that is fragile with respect to the behaviour of the environment.

The problem could be dismissed as merely another example of over-fitting, except that the type of over-fitting is highly specific, and occurs due to confounding controllable mechanisms with the uncontrollable environment. Avoiding the problem requires a method of specifying, or learning, the distinction.

Asada et al. (1998) addressed a related issue. They described the differences, as relevant to reinforcement learning, between the self body and static environment; passive agents (e.g. moving target objects); and other, active agents (e.g. other robots or people). They show that reinforcement learning's treatment of other active agents is inadequate. Based on our results, we suggest that even passive agents can be a serious problem.

7.8 Suggestions for Further Work

A system that could learn the distinction between controllable mechanisms from uncontrollable enviroment would both simplify the learning problem and lead to more robust solutions. Failing that, other mechanisms that try to avoid learning an overly specific implicit dynamic model would be helpful. The learning system's model is a result not only of its experiences, but also of its state representation. If the state representation does not contain enough information then the learning system's generalisation from experiences will be misguided.

In the experiments reported in this chapter the target's acceleration was not included in the state representation. This could be seen as a deficiency since including that information in the state representation could have allowed the learning system to find a more general representation and avoided over specialisation. Still, it would not guarantee that the system of avoids learning specific, fragile relationships. Further, estimating acceleration is problematic in a low rate system (15Hz in this case). The estimate of acceleration would be lagged and noisy.

Another problem with including the acceleration is increased state space dimensionality. Although algorithms that represent continuous functions scale better than those that discretise, the number of state variables that is feasible will always be limited. The WFNN learning system has not coped well with large state spaces, in particular the multiple degree of freedom experiments in this chapter and the wandering experiment in the previous chapter. This indicates that expanding the state space further is problematic. A deeper problem in the representation is that all pertinent data must be included in the *current* state. The learning system cannot exploit patterns in the state over time. The simple approach of including a sequence of states as the state information further inflates the size of the state space so that the problem becomes intractable.

The one degree of freedom visual servoing task was split between fixation to static targets, and pursuit of a swinging target, resulting in improved performance on both tasks. No attempt was made to learn the division of labour, or even to automatically switch between controllers. Divide and conquer approaches are common, pragmatic methods for improving learning performance. However, the process must be more highly automated for learning to be worthwhile.

The most practical path for research seems to be new methods of incorporating existing knowledge into the controller to reduce the learning problem to manageable proportions, and to reduce the reliance on active learning. Simple approaches to incorporating existing knowledge have included separating the degrees of freedom and building on existing controllers or biologically inspired models but there is a need for a systematic, practical approach to building in knowledge without overly limiting the learning system's potential to develop an innovative controller.

7.9 Summary

This chapter presented the first continuous state or continuous state and action reinforcement learning controller for an active head. The controller produced smooth actions, even in response to sudden changes in the state. Use of target velocity information was an important feature, based on the evidence from biological systems and previous controllers.

Experiments using the active head were more convenient than the mobile robot experiments described in the previous chapter because the active head's environment was more constrained and it was easier to maintain safety. Despite this, it was still laborious to provide experiences to the learning system. A training apparatus reduced the labour involved but was only capable of producing stereotypical movements.

The single degree of freedom configuration was stable and allowed quantitative experimentation. The controller performed lag-free tracking of a swinging target. It was possible through implicitly predicting the target's behaviour. Reinforcement learning's ability to optimise behaviour over time helps to compensate for sensing delays. Although the lag-free tracking performance was excellent, the controller's solution was somewhat fragile with respect to changes in the target's behaviour. The model-free approach is the source of the fragility: it makes no distinction between controllable mechanisms and uncontrollable environment.

The four degrees of freedom controller could be regarded as the first multiple degree of freedom (without separating degrees of freedom) learned controller for an active head; However, the controller failed at times to consistently control all degrees of freedom simultaneously. The instability of the multiple degree of freedom controller led to a system of resetting the neural network to random weights when controller divergence was detected. The controller then learnt passively from already gathered experiences while learning actively through interaction with the environment. The learning system learnt quickly when a large database of experiences was available, demonstrating the strong potential of the passive learning ideas described in Chapter 2.

Chapter 8

Conclusions

A monk asked, "What is one pertinent statement?"
The master said, "You've talked it to death."

Joshu, c. 890, no. 294

This thesis is a contribution toward the practical use of reinforcement learning methods for robotics. A continuous state and action reinforcement learning algorithm was applied to two robots, resulting in novel behaviour. Promising preliminary investigation of the passive learning concept was also carried out. This chapter summarises what was achieved and suggests directions for further research.

8.1 Summary

Robots can learn from experience through reinforcement learning techniques. Chapter 2 introduced the reinforcement learning framework and the $\mathcal{Q}$-Learning method for solving reinforcement learning problems. $\mathcal{Q}$-Learning systems can learn from off-policy actions—actions that do not follow the current best-known strategy. This gives it the ability to learn from experience replay, other controllers, other behaviours, phantom targets, and data transformations. These abilities are important for robotics as they can reduce experimentation time.

$\mathcal{Q}$-Learning is conventionally applied to discrete state and action problems. However, robotics problems usually include continuous state and action variables. Requirements for continuous state and action $\mathcal{Q}$-learning were proposed in Chapter 2. Existing algorithms were described and evaluated with reference to these requirements.

Chapter 3 described our WFNN continuous state and action $\mathcal{Q}$-learning method. The properties of the algorithm were investigated in the following chapter and several stability problems were discussed. In particular, we discovered a link between the rising $\mathcal{Q}$ problem (Thrun and Schwartz, 1993) and the advantage learning algorithm (Harmon and Baird, 1995). Visualising the action-values and neural network weights with graphs was a valuable aid in investigating convergence problems. The WFNN method was not stable enough to solve a pure delayed reward problem. However, we argued that it does not imply that the WFNN is not useful for robotics.

Learning on real robots is different to learning in simulation. Apart from sensitivity to errors in sensing and timing, safety management is a problem. Vetoing dangerous actions and filtering in the interests of safety can trivialise the learning task so that solution of the task is inevitable, but uninteresting. Conversely, safety mechanisms can interfere with learning and make solution of the task impossible. Chapter 5 proposed architectures that reduce the conflict between safety monitoring and learning.

Chapter 5 also proposed architectures that reduce timing errors and maximise the available processing time. Our approach was to separate any-time tasks from real-time tasks, ensure scalability, and consider the effects of data transmission errors.

The WFNN method was applied to two mobile robotics tasks in Chapter 6. In the first task, the robot learnt to smoothly pursue targets using vision. After learning, the robot moved smoothly and was able to pursue targets without calibration. The second task was to *wander*; this involved moving and avoiding obstacles. A novel reward function was developed for the task: higher rewards for moving forwards and viewing open spaces. The experiments showed the importance of choosing suitable state and action representations and reward function. The state representation chosen for the wandering behaviour was inappropriate, the state space was too large and the representation and did not exploit the generalisation properties of the learning algorithm. The passive learning approach was demonstrated through *learning from other behaviours*. The pursuit behaviour learnt by observing the wandering behaviour and vice versa.

Learning algorithms are often suggested as a tool for improving the robustness of robot behaviour. However, we found that the process of learning has hidden robustness problems compared to a hard-wired controller. A non-learning control system can cope with intermittent sensing problems; whereas a learning system can fail to separate the false sensor readings from the true and be misled. At a minimum, the data must be checked for sanity to avoid disrupting learning algorithms.

We found that if smooth actions are desired it is necessary to build the desire into the reward function through energy and coarseness penalties. In contrast, some researchers have obtained smooth actions without reward penalties. The difference in the algorithms appears to be that the WFNN algorithm is capable of producing actions that do not vary smoothly; whereas some other algorithms cannot. Rather than being hard-wired to produce smoothly varying actions, the WFNN algorithm learnt to make a tradeoff between the requirements of the problem and produced smoothly varying actions as a result.

The slowness of the mobile robot made it difficult to assess whether the learning system was compensating for the speed of the target. The limited field of view was a problem for the wandering algorithm. Chapter 7 addressed these problems through experiments using an active vision head. In these tasks the learning system had to control the joints of the active head in order to keep a target centred in the camera view. A flexible software architecture allowed any combination of the four joints (pan, tilt, left verge axis, and right verge axis), and two cameras to be incorporated into the learning problem. The learning algorithm's stability decreased as more joints and cameras were added. To compensate for the instability a system of saving and loading experience was developed. The learning system learnt quickly from the reloaded experiences, further demonstrating the strong potential of the passive learning concept.

The active head experiments showed conclusively that the learning algorithm could compensate for the speed of the target, and further, that it can learn behavioural patterns. In this case it learnt the characteristic pendulum swing and was able to perform lag-free tracking through motion prediction. This demonstrated that the WFNN algorithm is capable of solving delayed reward problems and producing smooth, high accuracy movements.

Although the learnt pendulum model was implicit in the controller, it was possible to extract some aspects of the model. The method of interpreting the implicit model through measuring the response of the controller to different inputs allowed us to gain a greater understanding of the learned controller. We compared the extracted model to a theoretically derived pendulum model and found them to be similar. Implicit modelling was found to have drawbacks: a fragile controller can result due to the failure to separate the controllable from the non-controllable.

Our continuous state and action $\mathcal{Q}$-learning algorithm was successful in simulation, but more importantly, it was successful in the real world. It was capable of learning a variety of behaviours on two robotic platforms in reasonable time, and produced smooth, controlled movements. The experiments led us to a deeper understanding of the problems of applying learning systems to robots.

8.2 Future Directions

Continuous state and action reinforcement learning does not have the convergence guarantees of discrete state and action reinforcement learning. This does not imply that continuous state and action methods are impractical or not worthy of further study. Continuous methods broaden the range of problems that can be tackled. Besides, the convergence guarantees of discrete state and action reinforcement learning do not carry beyond simulation, primarily due to unrealistic assumptions.

Continuous state and action reinforcement learning is worthwhile; smooth, accurate movement is possible and practical. The WFNN is just one implementation of continuous state and action reinforcement learning; a simpler, more transparent, and more stable algorithm would be desirable. Moving towards a continuous time framework and $\mathcal{Q}(\lambda)$ learning may also be worthwhile. This thesis has suggested requirements for continuous state and action $\mathcal{Q}$-learning algorithms. The requirements are motivated by the needs of robotics, rather than the needs of simulations.

The simulation results presented in the earlier chapters required months of computer simulation time, but the human effort was small once the algorithm had been coded properly. In contrast, the robot experiments took a few weeks but induced considerable strain. Apart from the work for the people taking turns monitoring the robot, the concentration of other lab members was impeded due to the robot moving unpredictably around their working environment. Safety management was a critical problem: long learning time, repeated experiments, and unpredictable behaviour induced boredom but required vigilance. One implication for learning systems on robots is that a quantum leap in reinforcement learning algorithms is required in order to apply these techniques to robots. An alternative, and more practical, conclusion is that a methodological change is needed.

The robot experiments in this thesis followed the tabula rasa approach, learning from scratch. It results in slow learning, and worse, re-learning what is already known. Failure to build in any information can also lead to learning strategies that are fragile, as in the active head experiments reported in Chapter 7. Learning in simulation or model-based control avoids the tabula rasa approach, but heavily restricts the controllers that can be learnt and can prevent development of novel behaviour.

The passive learning approach builds on existing knowledge in a different way. It partially develops a controller based on data gathered on the real robot. Designing one or more rough controllers that perform some task is currently easier and safer than persuading a learning system to learn a particular task, even if the learning system has already been debugged in simulation. Additionally, some controller is almost always available already when learning research begins. Data can be gathered easily using existing controllers because behaviour is fairly predictable. A controller learnt from the data can be refined online through active learning. Passive learning and other approaches to using existing knowledge are suggested as strong focuses for future research. The most important consideration is that off-line processing time is free, on-line processing time is inexpensive, and experimenter's time is precious.

> *The old [robot vision] paradigm was a natural child of the hippy era—visual systems were supposed to relax and take in the scene.*
>
> *The new [active vision] paradigm is born of the eighties, a robot yuppy that makes straight for its goal, immune to distraction.*
>
> *Blake and Yuille, 1992, Preface*

We hope that robots will become more flexible than today's robot yuppy, and be capable of learning from their mistakes and the experience of others.

Appendix A

Example Interface for Dynamic-System Simulators

Various interfaces to the WFNN learning system that are suitable for real-time learning were described in Chapter 5. The interface designs reflected their implementation in procedural languages (e.g. C++, Fortran). Control systems are often evaluated using a non-procedural environment.

This appendix presents an example interface for the WFNN algorithm for the Simulink® [1] graphical environment for simulating dynamic systems (MathWorks, 2000). Developing a Simulink interface for a learning algorithm makes comparison to standard control algorithms easier, and could broaden a learning algorithm's accessibility and appeal.

Figure A.1 shows the WFNN learning system in a Simulink model. The interface is wrapped around the *act-learn-think* architecture. However, since it has been implemented as a single Simulink block the interface appears similar to the *learn* architecture. A *don't learn* input is provided so that the internal call to the `learn` method can be disabled. The `thinks` variable (the number of experience replays) can be adjusted in a dialog box by double clicking on the WFNN block. The Simulink interface does not support off-policy learning techniques because it does not allow direct setting of the action and next-state. The action can be modified through the *exploration* input. This input is added to the action internally. The *value* output produces the expected value of the input state, $\max_u \mathcal{Q}(x, u)$, not the expected value of the output action in the input state, $\mathcal{Q}(x, u)$. (These values can be different because of the exploration input.)

The example task is to position an object at a target position through velocity control

[1]Simulink is registered trademark of The MathWorks, Inc.

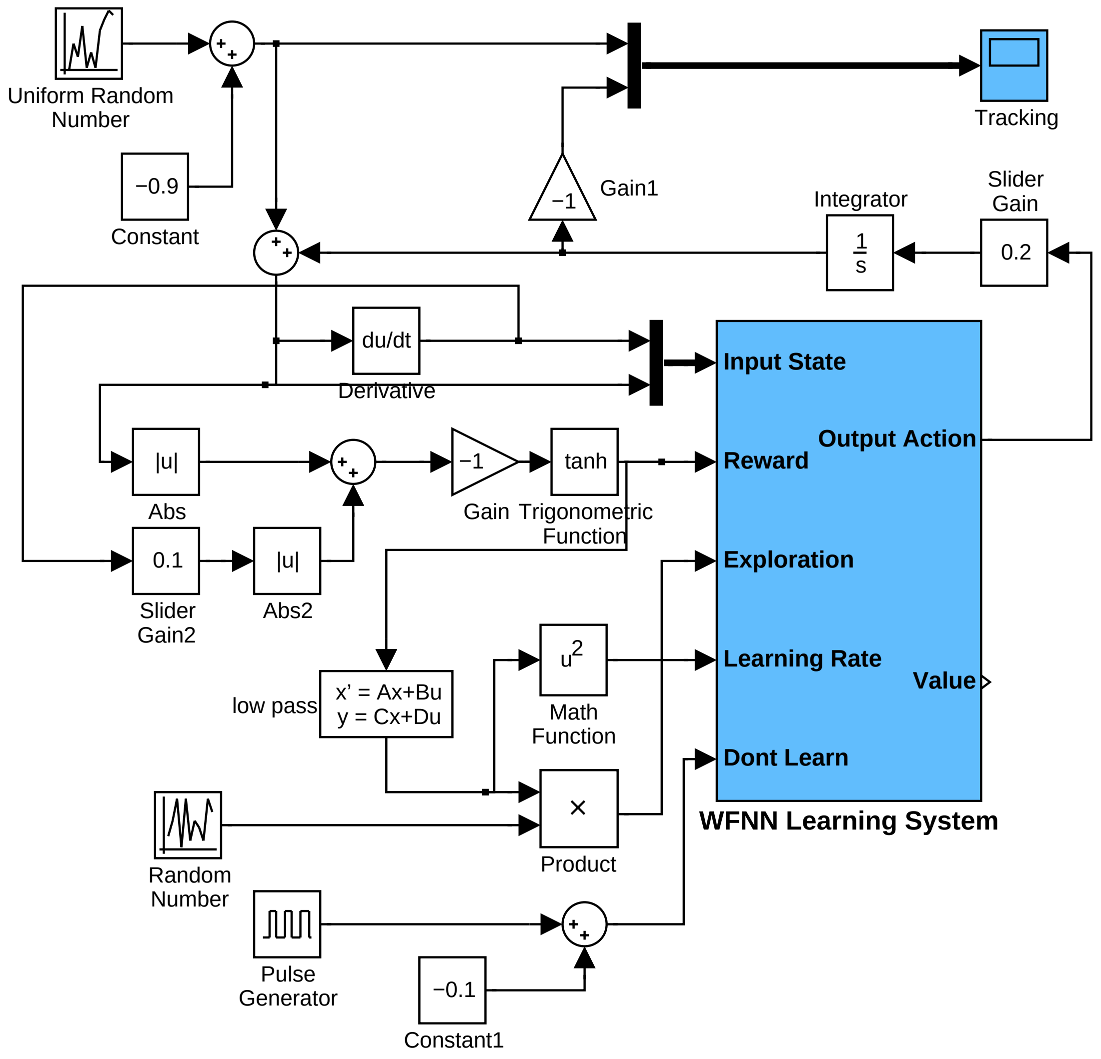


Figure A.1: Use of the WFNN algorithm within Simulink

with one degree of freedom. The dynamics of the system are simple:

$$\frac{dp}{dt} = 0.2u$$

where p is the position of the object; and u is the velocity, controlled directly by the learning system. This is shown graphically in the diagram. The state is composed of the error in the position of the object and the velocity of the object. The reward has components from the positioning error and velocity. The learning rate is controlled by the filtered value of the reward. Again, these details are shown in the diagram.

The desired position of the object changes occasionally due to the random number generator in the upper left corner of the diagram. While this change of desired position is occurring the *don't learn* input is activated by the pulse generator in the lower left corner. This is an example of ignoring irrelevant state transitions.

A sample run of the learning system is shown in Figure A.2. Traces show the target position and the actual position. The graph was produced by block marked 'tracking' in the upper right corner of the diagram.

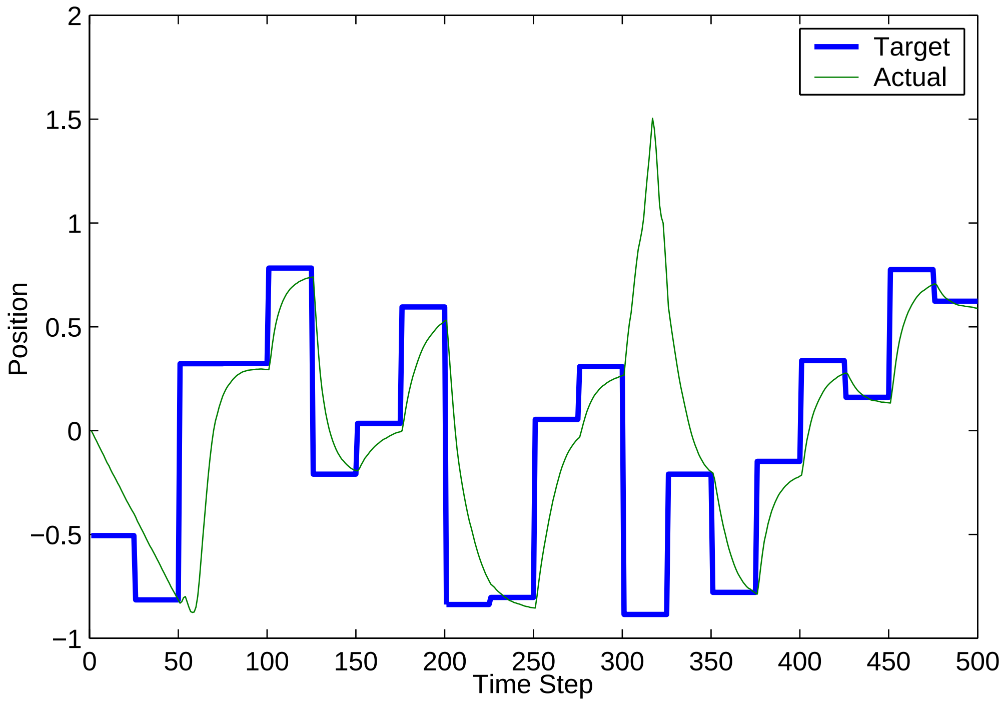


Figure A.2: Graph produced by the 'tracking' block in Figure A.1 showing learning to control position. Learning begins from scratch at time 0. At 300 time steps the learning system appears to be degrading but it recovers at around 320 time steps.

Appendix B

Wire Fitting C++ Source Code

This source code implements step 3b from the WFNN algorithm in Chapter 3. This includes evaluation of Equations (3.1), (3.2), (3.3), (3.4), and (3.5). An electronic version of the source code is included on the CD-ROM that accompanies this thesis. It can also be downloaded from the author's website.

```
class Wirefit
{
public:
  Wirefit(int no_actions, int no_wires, double smoothing);
  ~Wirefit();
  /** calculate new wire values so Q(chosen_action) moves toward Qnew

    all arrays must be provided by caller<p>
    actions array flattened 2d, should be no_wires rows, no_actions columns
    so a wire can be accessed as a chunk (row) of the array<p>
    result placed in new_actions, new_values
    @returns old Q
    */
  double learn(const double * const actions,
               const double * const chosen_action,
               const double * const values,
               double * const new_actions,
               double * const new_values,
               double Qnew);
protected:
  // vectors must be m_no_wires long
  // returns sum of squared differences
  double vdist(const double * const d1, const double * const d2);
  double smoothdist(const double * const u1, const double * const u2,
                    double y, double ymax);
  double maximum(const double * const d);
protected:
  const int m_no_actions;
  const int m_no_wires;
  const double m_smoothing;
  static const double m_eps = 0.0001;
};

Wirefit::Wirefit(int no_actions, int no_wires, double smoothing)
```

```
  : m_no_actions(no_actions), m_no_wires(no_wires),
    m_smoothing(smoothing)
{}

Wirefit::~Wirefit() {}

// returns old Q
double Wirefit::learn(const double * const actions,
                      const double * const chosen_action,
                      const double * const values,
                      double * const new_actions,
                      double * const new_values,
                      double Qnew)
{
  double wsum = 0.0;
  double nrm = 0.0;
  double ymax = maximum(values);
  double pddenom;
  double pdxgeneric;
  double smdist[m_no_wires];
  double Q;
  double yd[m_no_wires];
  double ud[m_no_wires * m_no_actions];
  // sum of squares of partial derivatives
  double totalyd = 0.0, totalud = 0.0;
  int i;

  // calculate nrm, wsum and Q
  for (i = 0; i < m_no_wires; i++) {
    smdist[i] = smoothdist(actions+i*m_no_actions, chosen_action,
                           values[i], ymax);
    wsum += values[i]/smdist[i];
    nrm += 1.0/smdist[i];
  }
  Q = wsum/nrm;

  // find partial derivatives in terms of the values (y)
  // and the actions (u)
  for (i = 0; i < m_no_wires; i++) {
    pddenom = nrm*smdist[i];
    pddenom *= pddenom;
    yd[i] = (nrm*(smdist[i]+values[i]*m_smoothing)
             - wsum*m_smoothing)
      / pddenom ;
    totalyd += yd[i]*yd[i];
    pdxgeneric = (wsum-nrm*values[i])*2.0/pddenom;
    for (int j = 0; j < m_no_actions; j++) {
      ud[i*m_no_actions + j] =
        pdxgeneric*(actions[i*m_no_actions + j]-chosen_action[j]);
      totalud += ud[i*m_no_actions + j]*ud[i*m_no_actions + j];
    }
  }

  for (i = 0; i < m_no_wires; i++) {
    new_values[i] = values[i] + (Qnew - Q) * yd[i];
    for (int j = 0; j < m_no_actions; j++) {
      new_actions[i*m_no_actions + j] = actions[i*m_no_actions + j]
        + (Qnew - Q) * ud[i*m_no_actions + j];
    }
  }
  return Q; // old Q
}

// sum of squared differences
double Wirefit::vdist(const double * const d1,
                      const double * const d2)
{
  double dist = 0.0;
  double tmp;
  for (int i=0; i < m_no_actions; i++) {
    tmp = d1[i] - d2[i];
    dist += tmp*tmp;
  }
  return dist;
}

double Wirefit::smoothdist(const double * const u1,
                           const double * const u2,
                           double y, double ymax)
{
  return vdist(u1,u2) + m_smoothing*(ymax - y) + m_eps;
}

double Wirefit::maximum(const double * const d)
{
  double m = d[0];
  for (int i = 1; i < m_no_wires; i++)
    if (d[i] > m) m = d[i];
  return m;
}
```

References

Albus, J. S. (1975), A new approach to manipulator control: the cerebrellar model articulated controller (CMAC), *Journal of Dynamic Systems, Measurement and Control* **97**:220–227.

Aloimonos, Y. (1990), Purposive and qualitative active vision, *in Proceedings of the Image Understanding Workshop*.

Aloimonos, Y., Weiss, I., and Bandopadhay, A. (1988), Active vision, *International Journal of Computer Vision* **7**:333–356.

Anderson, B. (1985), Adaptive systems, lack of persistency of exitation and bursting phenomena, *Automatica* **21**.

Anderson, C. W. (1998), Pole-balancer video game Java applet, `http://www.cs.colostate.edu/∼anderson/code/polejava.html`.

Asada, M., Hosoda, K., and Suzuki, S. (1998), Vision-based learning and development for emergence of robot behaviors, *in* Shirai, Y. and Hirose, S., editors, *Robotics Research, The Seventh International Symosium*, pp. 327–338, Springer.

Asada, M., Uchibe, E., Noda, S., Tawaratsumida, S., and Hosoda, K. (1994), Vision-based behavior acquisition for a shooting robot by using a reinforcement learning, *in Proceedings of the IAPR/IEEE Workshop on Visual Behaviors*.

Aslin, R. N. (1981), Development of smooth pursuit in human infants, *in Proceedings of the Last Whole Earth Eye Movement Conference*, Florida.

Bagnell, J. A. and Schneider, J. G. (2001), Autonomous helicopter control using reinforcement learning policy search methods, *in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA2001)*, Seoul, Korea.

Baird, L. C. (1994), Reinforcement learning in continuous time: Advantage updating, *in Proceedings of the International Conference on Neural Networks*, Orlando, Florida.

Baird, L. C. (1995), Residual algorithms: Reinforcement learning with function approximation, *in Proceedings of the 12th International Conference on Machine Learning*, San Francisco.

Baird, L. C. and Klopf, A. H. (1993), Reinforcement learning with high-dimensional, continuous actions, Technical Report WL-TR-93-1147, Wright Laboratory.

Bajcsy, R. and Allen, P. (1984), Sensing strategies, *in Proceedings of the U.S.-France Robotics Workshop*, Philadelphia, PA.

Bajcsy, R. and Campos, M. (1992), Active and exploratory perception, *CVGIP: Image Understanding* **56**(1).

Baker, W. and Farrel, J. (1992), An introduction to connectionist learning control systems, *in* White and Sofge (1992).

Ballard, D. H. (1991), Animate vision, *Artificial Intelligence* **48**:57–86.

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

Ballard, D. H. and Brown, C. M. (1992), Principles of animate vision, *CVGIP: Image Understanding* **56**(1).

Barrow, H. G. and Tenenbaum, J. M. (1981), Interpreting line drawings as three-dimensional surfaces, *Artificial Intelligence* **17**:75–116.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983), Neuronlike adaptive elements that can solve difficult learning control problems, *IEEE Transactions on systems, man and cybernetics* **SMC-13**:834–846.

Baxter, J., Weaver, L., and Bartlett, P. L. (1999), Direct gradient-based reinforcement learning: Ii. gradient ascent algorithms and experiments, Technical report, Computer Sciences Laboratory, Australian National University.

Becker, W. (1991), Saccades, *in* Carpenter (1991a).

Bellman, R. (1957), *Dynamic Programming*, Princeton.

Benbrahim, H. and Franklin, J. A. (1997), Biped dynamic walking using reinforcement learning, *Robotics and Autonomous Systems* **22**:283–302.

Berenji, H. R. (1994), Fuzzy Q-learning: a new approach for fuzzy dynamic programming, *in Proceedings of the Third IEEE International Conference on Fuzzy Systems*, IEEE Computer Press, NJ.

Berthouze, L. and Kuniyoshi, Y. (1998), Emergence and categorization of coordinated visual behavior through embodied interaction, *Machine Learning* **31**.

Berthouze, L., Rougeaux, S., Kuniyoshi, Y., and Chavand, F. (1996), A learning stereo-head control system, *in Proceedings of the World Automation Congress/International Symposium on Robotics and Manufacturing*, France.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996), *Neuro-Dynamic Programming*, Athena scientific, Belmont, MA.

Bitmead, R. R., Gevers, M., and Wertz, V. (1990), *Adaptive Optimal Control: The Thinking Man's GPC*, Prentice Hall.

Blake, A. and Yuille, A., editors (1992), *Active Vision*, MIT Press.

Bonarini, A. (1996), Delayed reinforcement, fuzzy Q-learning and fuzzy logic controllers, *in* Herrera, F. and Verdegay, J. L., editors, *Genetic Algorithms and Soft Computing*, Studies in Fuzziness, 8, pp. 447–466, Physica-Verlag, Berlin, Germany.

Boyan, J. A. and Moore, A. W. (1995), generalization in reinforcement learning: safely approximating the value function, *in Proceedings of the neural information processing systems 7*.

Brooks, A., Dickins, G., Zelinsky, A., Keiffer, J., and Abdallah, S. (1997), A high-performance camera platform for real-time active vision, *in Proceedings of the First International Conference on Field and Service Robotics*, Canberra, Australia.

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

Brooks, R. A. (1991), Intelligence without reason, *in Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pp. 569–595, Sydney, Australia.

Brooks, R. A. and Mataric, M. J. (1993), Real robots, real learning problems, *in* Connell and Mahadevan (1993b).

Brown, C. (1990), Gaze controls with interactions and delays, *IEEE Transactions on Systems, Man, and Cybernetics* **20**(1):518–527.

Buessler, J., Urban, J., Kihl, H., and Gresser, J. (1996), A neural model for the visual servoing of a robotic manipulator, *in Proceedings of the Symposium on Control, Optimization and Supervision, Computational Engineering in Systems Applications (CESA'96)*, Lille, France.

Burt, P. J. (1988), Smart sensing in machine vision, *in* Freeman, H., editor, *Machine Vision: Algorithms, Architectures and Systems*, Academic Press, San Diego, CA.

Carpenter, R. H. S., editor (1991a), *Eye Movements*, volume 8 of *Vision and Visual Dysfunction*, Macmillan.

Carpenter, R. H. S. (1991b), The visual origins of ocular motility, *in* Carpenter (1991a).

Cheng, G. and Zelinsky, A. (1996), Real-time visual behaviours for navigating a mobile robot, *in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'96)*, Osaka, Japan.

Cheng, G. and Zelinsky, A. (1998), Goal-oriented behaviour-based visual navigation, *in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '98)*, Leuven, Belgium.

Clouse, J. A. (1996), *On Integrating Apprentice Learning and Reinforcement Learning*, Ph.D. thesis, Computer Science, University of Massachusetts Amherst.

Collewijn, H. (1991), The optokinetic contribution, *in* Carpenter (1991a).

Collewijn, H., Martins, A. J., and Steinman, R. M. (1983), Compensatory eye movements during active and passive head movements: Fast adaption to changes in visual magnification, *Journal of Physiology* **340**:259–286.

Connell, J. H. and Mahadevan, S. (1993a), Rapid task learning for real robots, *in* Connell and Mahadevan (1993b).

Connell, J. H. and Mahadevan, S., editors (1993b), *Robot Learning*, Kluwer.

Conover, W. J. (1999), *Practical Nonparametric Statistics*, Wiley, third edition.

Dawkins, R. (1996), *Climbing Mount Improbable*, Viking, Great Britain.

DeMarco, T. (1978), *Structured Analysis and System Specification*, Prentice Hall/Yourdan.

Dietterich, T. G. (2000), Hierarchical reinforcement learning with the MAXQ value function decomposition, *Journal of Artificial Intelligence Research* **13**:227–303.

Doya, K. (1995), Temporal difference learning in continuous time and space, *in Proceedings of the Advances in Neural Information Processing Systems*.

Franz, M. and Mallot, H. (2000), Biomimetic robot navigation, *Robotics and Autonomous Systems* **30**:133–153.

Giles, C. L. and Lawrence, S. (1997), Presenting and analyzing the results of AI experiments: Data averaging and data snooping, *in Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI-97*, pp. 362–367, AAAI Press, Menlo Park, California.

Glorennec, P. Y. (1994), Fuzzy Q-learning and evolutionary strataegy for adaptive fuzzy control, *in Proceedings of the European Congress on Intelligent Techniques and Soft Computing (EUFIT'94)*, ELITE Foundation, Aachen, Germany.

Gordon, G. J. (1999), *Approximate Solutions to Markov Decision Processes*, Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Gorse, D., Shepherd, A., and Taylor, J. G. (1997), The new ERA in supervised learning, *Neural Networks* **10**:343–352.

Gross, H.-M., Stephan, V., and Krabbes, M. (1998), A neural field approach to topological reinforcement learning in continuous action spaces, *in Proceedings of the IEEE World Congress on Computational Intelligence, WCCI'98 and International Joint Conference on Neural Networks, IJCNN'98*, Anchorage, Alaska.

Gullapalli, V. (1990), A stochastic reinforcement learning algorithm for learning real-valued functions, *Neural Networks* pp. 671–692.

Gupta, M. M. and Rao, D. H., editors (1994), *Neuro-Control Systems: Theory and Applications*, IEEE Press.

Harmon, M. E. and Baird, L. C. (1995), Residual advantage learning applied to a differential game, *in Proceedings of the International Conference on Neural Networks*, Washington D.C.

Hashimoto, H., Kubota, T., Kudou, M., and Harashima, F. (1991), Self-organizing visual servo system based on neural networks, *in Proceedings of the American Control Conference*, Boston.

Heinzmann, J. and Zelinsky, A. (1997), Robust real-time face tracking and gesture recognition, *in Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI'97*.

Heller, D. (1988), On the history of eye movement recording, *in* Lüer et al. (1988).

Horswill, I. (1994), Visual collision avoidance by segmentation, *in Proceedings of the Intelligent Robots and Systems (IROS'94)*, pp. 902–909.

Hosoda, K. and Asada, M. (1994), Versatile visual servoing without knowledge of true jacobian, *in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*.

Hunt, K. J., Sbarbaro, D., Zbikowski, R., and Gawthrop, P. J. (1994), Neural networks for control systems — a survey, *in* Gupta and Rao (1994).

Hutchinson, S., Hager, G. D., and Corke, P. I. (1996), A tutorial on visual servo control, *IEEE Transactions on Robotics and Automation* **12(5)**:651–670.

Inaba, M. (1993), Remote-brained robotics: Interfacing AI with real world behaviors, *in Proceedings of Robotics Research: The Sixth International Symposium*, Hidden Valley.

Jones, G. M. (1991), The vestibular contribution, *in* Carpenter (1991a).

Jordan, M. and Jacobs, R. (1990), Learning to control an unstable system with forward modeling, *in Proceedings of the Advances in Neural Information Processing Systems 2*, San Mateo, CA.

Joshu, J. (c. 890), *The Recorded Sayings of Zen Master Joshu*, this edition Shambhala Publications 1998, translated by James Green.

Judge, S. J. (1991), Vergence, *in* Carpenter (1991a).

Jung, D., Cheng, G., and Zelinsky, A. (1997), Robot cleaning: an application of distributed planning and real-time vision, *in Proceedings of the International Conference on Field and Service Robotics (FSR'97)*, Canberra, Australia.

Jürgens, R., KornHuber, A. W., and Becker, W. (1988), Prediction and strategy in human smooth pursuit eye movements, *in* Lüer et al. (1988).

Kaelbling, L. P. (1993a), Hierarchical reinforcement learning: Preliminary results, *in Proceedings of the 10th International Conference on Machine Learning*.

Kaelbling, L. P. (1993b), Learning to achieve goals, *in Proceedings of the 13th International Joint Conference on Artificial Intelligence*.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996), Reinforcement learning: A survey, *Journal of Artificial Intelligence Research* **4**:237–285.

Kawato, M., Furawaka, K., and Suzuki, R. (1987), A hierarchical neural network model for the control and learning of voluntary movements, *Biological Cybernetics* .

Kimura, H. and Kobayashi, S. (1998), Reinforcement learning for continuous action using stochastic gradient ascent, *in Proceedings of the The 5th International Conference on Intelligent Autonomous Systems (IAS-5)*.

Kohonen, T. (1989), *Self-Organization and Associative Memory*, Springer, Berlin, third edition, [First edition, 1984].

Lancaster, P. and Šalkauskas, K. (1986), *Curve and Surface Fitting, an Introduction*, Academic Press.

Lawrence, S. and Giles, C. L. (2000), Overfitting and neural networks: Conjugate gradient and backpropagation, *in Proceedings of the IEEE International Conference on Neural Networks*, pp. 114–119, IEEE Press.

Leigh, R. J. and Zee, D. S. (1991), Oculomotor disorders, *in* Carpenter (1991a).

Lin, L.-J. (1992), Self-improving reactive agents based on reinforcement learning, planning and teaching, *Machine Learning* **8**(3/4).

Lo, W. C. (1996), Robotic visual servo control, *in Proceedings of the 12th International Conference on CAD/CAM, Robotics and Factories of the Future*, London.

Lorigo, L., Brooks, R., and Grimson, W. (1997), Visually-guided obstacle avoidance in unstructured environments, *in Proceedings of the Intelligent Robots and Systems (IROS'97)*, pp. 373–379, Grenoble, France.

Lüer, G., Lass, U., and Shallo-Hoffman, J., editors (1988), *Eye Movement Research: Physiological and Psychological Aspects*, C. J. Hogrefe, Göttingen, Germany.

Maire, F. (2000), Bicephal reinforcement learning, *in Proceedings of the 7th International Conference on Neural Information Processing (ICONIP-2000)*, Taejon, Korea.

Marjanović, M., Scassellati, B., and Williamson, M. (1996), Self-taught visually guided pointing for a humanoid robot, *in Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, MA.

Marr, D. (1982), *Vision*, W.H. Freeman.

Mataric, M. J. (1994), Reward functions for accelerated learning, *in Proceedings of the International Conference on Machine Learning*.

MathWorks (2000), *Using Simulink*, version 4 edition.

Metta, G., Panerai, F., Manzotti, R., and Sandini, G. (2000), Babybot: an artificial developing robotic agent, *in Proceedings of From Animals to Animats: Sixth International Conference on the Simulation of Adaptive Behavior (SAB 2000)*, Paris.

Michie, D. and Chambers, R. A. (1968), BOXES: An experiment in adaptive control, *in* Dale, E. and Michie, D., editors, *Machine Intelligence 2*, Oliver & Boyd, Edinburgh.

Moody, J. and Darken, C. J. (1989), Fast learning in networks of locally-tuned processing units, *Neural Computation* **1**:281–294.

Moravec, H. P. (1990), The Stanford Cart and the CMU Rover, *in* Cox, I. J. and Wilfong, G. T., editors, *Autonomous Robot Vehicles*, pp. 407–419, Springer-Verlag.

Munro, P. (1987), A dual back-propagation scheme for scalar reward learning, *in Program of the Ninth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ.

Murray, D. W., Bradshaw, K. J., McLauchlan, P. F., Reid, I. D., and Sharkey, P. M. (1995), Driving saccade to pursuit using image motion, *International Journal of Computer Vision* **16**(3):205–228.

Murray, D. W., Du, F., McLauchlan, P. F., Reid, I. D., Sharkey, P. M., and Brady, J. M. (1992), Design of stereo heads, *in* Blake and Yuille (1992).

Narendra, K. S. and Parthasarthy, K. (1994), Identification and control of dynamical systems using neural networks, *in* Gupta and Rao (1994).

Newell, A. and Simon, H. A. (1963), GPS, a program that simulates human thought, *in* Feigenbaum, E. A. and Feldman, J., editors, *Computers and Thought*, McGraw Hill.

Ng, A. Y. and Jordan, M. (2000), PEGASUS:A policy search method for large MDPs and POMDPs, *in Proceedings of the Uncertainty in Artificial Intelligence (UAI-2000)*, pp. 406–415, California.

Nguyen, D. and Widrow, B. (1990), The truck backer-upper: an example of self-learning in neural networks, *in* Miller, W. T., Sutton, R. S., and Werbos, P. J., editors, *Neural Networks for Control*, MIT Press, Cambridge, Mass.

Nilsson, N. J. (1984), SHAKEY the robot, Technical Report 323, SRI International, Menlo Park, California.

Ormoneit, D. and Sen, Ś. (1999), Kernel-based reinforcement learning, Technical Report 1999-8, Department of Statistics, Stanford University, CA.

Pagel, M., Maël, E., and von der Malsburg, C. (1998), Self calibration of the fixation movement of a stereo camera head, *Machine Learning* **31**(1-3):169–186.

Panerai, F., Metta, G., and G.Sandini (2000), Learning VOR-like stabilization reflexes in robots, *in Proceedings of the 8th European Symposium on Artificial Neural Networks (ESANN 2000)*, Bruges, Belgium.

Park, J. S. and Oh, S. Y. (1992), Dynamic visual servo control of robot manipulators using neural networks, *Journal of the Korean Institute of Telematics and Electronics* **29B**(10).

Pendrith, M. D. and Ryan, M. R. (1996), C-Trace: A new algorithm for reinforcement learning of robotic control, *in Proceedings of the ROBOLEARN-96*, Key West, Florida.

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

Piater, J. H., Grupen, R. A., and Ramamritham, K. (1999), Learning real-time stereo vergence control, *in Proceedings of the 14th International Symposium on Intelligent Control (ISIC '99)*.

Plaut, D. C., Nowlan, S. J., and Hinton, G. E. (1986), Experiments on learning by backpropagation, Technical Report CMU-CS-86-126, Computer Science Department, Carnegie-Mellon University, Pittsburgh.

Pola, J. and Wyatt, H. J. (1991), Smooth pursuit: Response characteristics, stimuli and mechanisms, *in* Carpenter (1991a).

Pomerleau, D. A. (1993), Knowledge-based training of artificial neural networks for autonomous robot driving, *in* Connell and Mahadevan (1993b).

Precup, D., Sutton, R. S., and Singh, S. (2000), Eligibility traces for off-policy policy evaluation, *in Proceedings of the 17th Conference on Machine Learning (ICML 2000)*, Morgan Kaufman.

Rao, R. P. and Ballard, D. H. (1994), Learning saccadic eye movements using multiscale spatial filters, *in Proceedings of Advances in Neural Information Processing Systems 7 (NIPS94)*.

Rashbass, C. (1961), The relationship between saccadic and smooth tracking eye movements, *Journal of Physiology* **159**:338–362.

Ritter, H., Martinetz, T., and Schulten, K. (1992), *Neural Computation and Self-Organizing Maps: An Introduction*, Addison Wesley.

Roberts, L. G. (1965), Machine perception of three-dimensional solids, *in* Tippet, J. T. et al., editors, *Optical and Electro-Optical Information Processing*, pp. 159–197, MIT Press, Cambridge, Massachusetts.

Rogers, K. (1978), The gambler, song from the album The Gambler.

Ruete, C. (1857), *Ein neues Ophthalmotrop, zur Erläuterung der Functionen der Muskeln und brechenden Medien des menschlichen Auges*, Teubner, Leipzig.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986), Learning internal representation by error propagation, *in* Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pp. 318–362, Cambridge, MA: MIT Press.

Rummery, G. and Niranjan, M. (1994), On-line Q-learning using connectionist systems, Technical Report CUED/F-INFEG/TR66, Cambridge University.

Rummery, G. A. (1995), *Problem solving with reinforcement learning*, Ph.D. thesis, Cambridge University.

Ryan, M. R. K. and Reid, M. (2000), Learning to fly: An application of hierarchical reinforcement learning, *in Proceedings of the 17th International Conference on Machine Learning (ICML-2000)*, pp. 807–814, CA.

Saito, F. and Fukuda, T. (1994), Learning architecture for real robot systems—extension of connectionist Q-learning for continuous robot control domain, *in Proceedings of the International Conference on Robotics and Automation (IROS'94)*, pp. 27–32.

Santamaria, J. C., Sutton, R. S., and Ram, A. (1998), Experiments with reinforcement learning in problems with continuous state and action spaces, *Adaptive Behaviour* **6**(2):163–218.

Santos, J. M. (1999), *Contribution to the study and design of reinforcement functions*, Ph.D. thesis, Universidad de Buenos Aires, Universite d'Aix-Marseille III.

Schraudolph, N. N. (1999), A fast, compact approximation of the exponential function, *Neural Computation* **11**(4).

Schraudolph, N. N. and Sejnowski, T. J. (1996), Tempering backpropagation networks: Not all weights are created equal, *in Advances in Neural Information Processing Systems 8*, MIT Press, Cambridge.

Sehad, S. and Touzet, C. (1994), Self-organising map for reinforcement learning: Obstacle avoidance with Khepera, *in Proceedings of Perception to Action*, Lausanne, Switzerland.

Shibata, T. and Schaal, S. (2001a), Biomimetic gaze stabilization based on feedback-error learning with nonparametric regression networks, *Neural Networks* **14**(2).

Shibata, T. and Schaal, S. (2001b), Biomimetic smooth pursuit based on fast learning of the target dynamics, *in Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS2001)*.

Smart, W. D. and Kaelbling, L. P. (2000), Practical reinforcement learning in continuous spaces, *in Proceedings of the 17th International Conference on Machine Learning*.

Srinivasan, M. and Venkatesh, S., editors (1997), *From Living Eyes to Seeing Machines*, Oxford University Press.

Sutherland, O., Rougeaux, S., Abdallah, S., and Zelinsky, A. (2000a), Tracking with hybrid-drive active vision, *in Proceedings of the Australian Conference on Robotics and Automation (ACRA2000)*, Melbourne, Australia.

Sutherland, O., Truong, H., Rougeaux, S., and Zelinsky, A. (2000b), Advancing active vision systems by improved design and control, *in Proceedings of the International Symposium on Experimental Robotics (ISER2000)*, Honolulu.

Sutton, R. (2001), Reinforcement learning FAQ: Frequently asked questions about reinforcement learning, web page.

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

Sutton, R. S. (1990), Integrated architectures for learning, planning, and reacting based on approximating dynamic programming, *in Proceedings of the 7th International Conference on Machine Learning*, pp. 216–224.

Sutton, R. S. and Barto, A. G. (1998), *Reinforcement Learning: An Introduction*, Bradford Books, MIT.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999a), Policy gradient methods for reinforcement learning with function approximation, *in Proceedings of the Advances in Neural Information Processing Systems 12*.

Sutton, R. S., Precup, D., and Singh, S. (1999b), Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning, *Artificial Intelligence* **112**:181–211.

Takahashi, Y., Takeda, M., and Asada, M. (1999), Continuous valued Q-learning for vision-guided behavior, *in Proceedings of the IEEE/SICE/RSJ International Conference on Multisensor Fusion and Integration for Intelligent Systems*.

Takeda, M., Nakamura, T., Imai, M., Ogasawara, T., and Asada, M. (2000), Enhanced continuous valued Q-learning for real autonomous robots, *in Proceedings of the International Conference of The Society for Adaptive Behavior*.

Tansley, K. (1965), *Vision in Vertebrates*, Chapman and Hall.

Terada, K., Takeda, H., and Nishida, T. (1998), An acquisition of the relation between vision and action using self-organizing map and reinforcement learning, *in Proceedings of the Second International Conference. Knowledge-Based Intelligent Electronic Systems (KES'98)*.

Tesauro, G. J. (1992), Practical issues in temporal difference learning, *Machine Learning* **8**(3/4):257–277.

Thrun, S. (1992), The role of exploration in learning control, *in* White and Sofge (1992).

Thrun, S. (1995), An approach to learning mobile robot navigation, *Robotics and Autonomous Systems* **15**(4):301–319.

Thrun, S. and Schwartz, A. (1993), Issues in using function approximation for reinforcement learning, *in Proceedings of the 1993 Connectionist Models Summer School*, Erblaum Associates, NJ.

Touzet, C. F. (1997), Neural reinforcement learning for behaviour synthesis, *Robotics and Autonomous Systems* **22**(3-4):251–81.

Truong, H. (2000), *Active Vision Head: A Novel Mechanism for Stereo Active Vision*, honours thesis, Robotic Systems Laboratory, Australian National University.

Tsai, R. (1987), A versatile camera calibration technique for high accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses, *IEEE Journal of Robotics and Automation* **RA-3**(4).

Tsitsiklis, J. N. and Roy, B. V. (1997), An analysis of temporal-difference learning with function approximation, *IEEE Transactions on Automatic Control* **42**:674–690.

von Helmholtz, H. (1962), Movements of the eyes, *in* Southall, J. P. C., editor, *Helmholtz's Treatise on Physiological Optics*, volume III, chapter 27, Dover Publications, New York, translation of the third German edition published in 1910.

Watkins, C. J. C. H. (1989), *Learning from Delayed Rewards*, Ph.D. thesis, University of Cambridge.

Watkins, C. J. C. H. and Dayan, P. (1992), Technical note: Q learning, *Machine Learning* **8**(3/4):279–292.

Werbos, P. J. (1992), Approximate dynamic programming for real-time control and neural modeling, *in* White and Sofge (1992).

White, D. A. and Sofge, D. A., editors (1992), *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, Van Nostrand Reinhold.

Whitehead, S. D. and Ballard, D. H. (1992), Learning to perceive and act by trial and error, *Machine Learning* **8**(3/4).

Williams, R. J. (1992), Simple statistical gradient-following algorithms for connectionist reinforcement learning, *Machine Learning* **8**:229–256.

Wilson, S. W. (1996), Explore/exploit strategies in autonomy, *in Proceedings of From Animals to Animats: Fourth International Conference on the Simulation of Adaptive Behavior (SAB 1996)*.

Wu, Q. and Stanley, K. (1997), Modular neural-visual servoing using a neuro-fuzzy decision network, *in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'97)*, Albuquerque, NM.

Wundt, W. (1862), Beschreibung eines künstlichen Augenmuskelsystems zur Untersuchung der Bewegungsgesetze des menschlichen Auges im gesunden und kranken Zustande, *Archiv für Opthalmologie* **8**(11):88–114.

Wyatt, J., Hoar, J., and Hayes, G. (1998), Design, analysis and comparison of robot learners, *Robotics and Autonomous Systems* **24**(1–2):17–32.

Yarbus, A. L. (1967), *Eye Movements and Vision*, Plenum Press, New York.

Yoshimi, B. and Allen, P. (1994), Active, uncalibrated visual servoing, *in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'94)*, San Diego, CA.

Zadeh, L. (1965), Fuzzy sets, *Information and Control* **8**:338–353.

Zhang, W. (1996), *Reinforcement Learning for Job-Shop Scheduling*, Ph.D. thesis, Oregon State University.

Zhao, D. L., Lasker, A. G., and Robinson, D. A. (1993), Interactions of simultaneous saccadic and pursuit prediction, *in Proceedings of Contemporary Ocular Motor and Vestibular Research: A Tribute to David A. Robinson*, pp. 171–180.

Index