Efficient Memory-based Learning for Robot Control

Andrew William Moore

Trinity Hall

A dissertation submitted for the degree of Doctor of Philosophy in the University of Cambridge.

October 1990

Abstract

This dissertation is about the application of machine learning to robot control. A system which has no initial model of the robot/world dynamics should be able to construct such a model using data received through its sensors—an approach which is formalized here as the *SAB* (State-Action-Behaviour) control cycle. A method of learning is presented in which all the experiences in the lifetime of the robot are explicitly remembered. The experiences are stored in a manner which permits fast recall of the closest previous experience to any new situation, thus permitting very quick predictions of the effects of proposed actions and, given a goal behaviour, permitting fast generation of a candidate action. The learning can take place in high-dimensional non-linear control spaces with real-valued ranges of variables. Furthermore, the method avoids a number of shortcomings of earlier learning methods in which the controller can become trapped in inadequate performance which does not improve. Also considered is how the system is made resistant to noisy inputs and how it adapts to environmental changes. A well founded mechanism for choosing actions is introduced which solves the experiment/perform dilemma for this domain with adequate computational efficiency, and with fast convergence to the goal behaviour. The dissertation explains in detail how the *SAB* control cycle can be integrated into both low and high complexity tasks. The methods and algorithms are evaluated with numerous experiments using both real and simulated robot domains. The final experiment also illustrates how a compound learning task can be structured into a hierarchy of simple learning tasks.

Acknowledgements

I would like to thank my supervisor William Clocksin for the initial motivation for this work, and for his help and advice. I am very grateful to Mary Lee and Thomas Vogel who provided valuable and detailed comments on many drafts of the dissertation. Thanks are also due to Thomas Clarke and Barney Pell for many useful and inspiring discussions. I am grateful to Chris Atkeson for introducing me to a number of important pieces of related work.

Declaration

I hereby declare that this dissertation is the result of my own work and, unless explicitly stated in the text, contains nothing which is an outcome of work done in collaboration. No part of this dissertation has already been or is currently being submitted for any degree, diploma or other qualification at any other university.

This dissertation is dedicated to my parents.

Contents

Chapter 1

Introduction

The introduction starts with a simple example of a robot control problem. Motivations for learning control are briefly reviewed, and there is a statement of those areas of the subject addressed by this work. Finally, some guidance is given about the structure of the rest of the dissertation.

This dissertation is about robots which can autonomously develop their own models of the world. Figure 1.1 shows a robot looking at its hand. At all times the controller must choose joint torques to cause the perceived position of the hand to behave in a way which helps achieve some task (such as moving towards the cross-hairs). Such a control problem can be solved if the robot knows how the following are related:

- The perceived **State**: the location of the image of the hand, and also its perceived speed and direction.
- The raw **Action**: the signals sent to the motors.
- The perceived **Behaviour**: the change in perceived position and velocity which then occurs.

This relationship is the composition of many other relationships. How does the torque at a joint depend on the signal to the joint? How does the torque affect the angular acceleration of the joint in this particular configuration? How does the configuration affect the perceived hand position?

This is an example of the central problem of low level robotics—the need to compute the relationship between a number of variables related to the robot's state and the environment. Such relations are termed *world models*. They include kinematic models, hand-eye coordination models, dynamic models, and spatial models. The models are difficult to obtain mathematically for reasons described later. A more appealing, and arguably more practical way to obtain them is through learning.

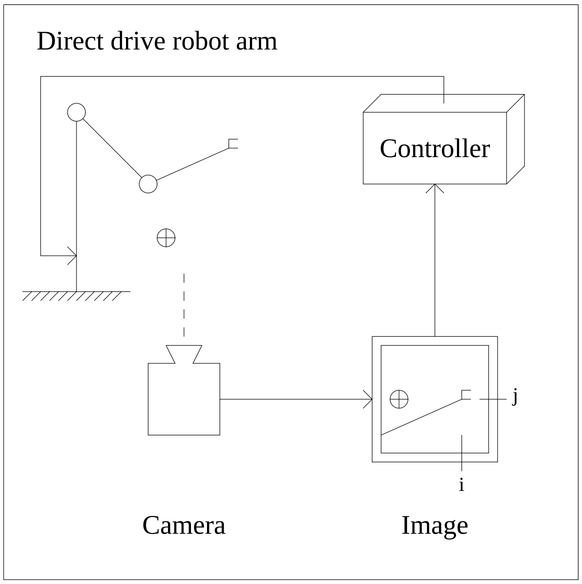

Figure 1.1

A robot looking at its own torque-controlled arm.

1.1 Learning Control: Motivations

A system which can improve itself is an aesthetically pleasing thing. A related motivation for designing learning robots is to understand how learning occurs in biological organisms. A successful automatic learning system might provide an indication as to how animals and people learn. Robot learning seems a particularly good place to begin, because motor learning is perhaps the most basic form of learning behaviour in organisms. It is not, however, guaranteed that engineering a solution provides a complete biological explanation. An analogy is flight, in which the engineered solution differs markedly from the biological solution.

However, the most important motivation is a practical one. Conventional control cannot cope with the sorts of interesting, autonomous machines which would be substantially different from present industrial machinery. The main reason is the difficulty of precoding a sufficiently general world model to accurately take account of all eventualities. The world is complex—even analytic models for simple components such as the relationship between joint angles, velocities, accelerations, and torques are very complex. This is true even when highly idealized and simplified component models are used.

1.2 The SAB Learning System

This dissertation is about a practical, efficient, fast and robust method to obtain robot world models by learning. The use of the word "robot" is for conciseness: the work is also applicable to other dynamic control problems which need multivariate models of the world. The method is called the *SAB Learning System*. The acronym denotes the three components of a dynamic world model:

state, action and behaviour. The principal aims of *SAB* learning are listed here:

- **Practical.** The work is strongly motivated by a desire to avoid the "Micro-world" problem. It is concerned with learning in complex high-dimensional state and control spaces with real-valued ranges of variables.
- **Efficient.** The time to update the world model with new knowledge and to use the model that is learned is sufficiently fast that it can realistically occur as the robot operates. This is attained by means of computationally efficient algorithms.
- **Fast.** The learning method is fast so that performance improves very quickly. This is achieved partly by means of a powerful generalization, but primarily by using a *one-shot* learning method: only one presentation of a piece of data is required for its information to be stored. It is not the case that something must be seen a number of times, each time perturbing the world model towards a representation which lessens the error.
- **Robust.** The learning method can cope with disorder in the environment, both in the form of noise, and in the form of either gradual change or sudden unpredictable change. It is also robust with respect to internal parameters, which can be chosen with minimal foreknowledge of the kind of relationship being learned. Finally, it is a method which is hard to get "stuck"—it will not repeat the same error.

This investigation also explains and demonstrates how learning world models can be sufficient to transform the design of robot controllers into simpler design problems. For simple controllers such as trajectory followers, or pick and place, there is almost no additional effort. For compound tasks, the use of learned world models can keep the controller design process at an entirely abstract level which renders the design problem easy for a human, and perhaps even automatable.

1.3 This Dissertation

This dissertation begins with an introduction to the techniques and problems of robot modelling and control, and then an introduction to both earlier and current work in the field of learning control. It formalizes the behaviour of a learned model-based controller and then discusses *how* world models might be represented. It then explains in detail why the chosen representation can be expected to meet the goals (in turn) of practicality, efficiency, speed, and robustness. The last two features require special attention and are dealt with in their own chapters.

By this time further issues have been uncovered, including the curse of dimensionality, and the search for a useful diversity of experience. These problems are explained and then dealt with using an algorithm called the *SAB Action Chooser*. Following this there is discussion of how best to use the world model to accomplish tasks.

After the main body of the dissertation, a variety of experiments are conducted to evaluate the method's performance. These experiments include

- Learning hand-eye coordination of a real five-jointed arm.
- Learning a visually observed trajectory of a simulated torque-controlled arm under a wide variety of conditions.
- Learning movement control over a wide variety of trajectories for the same arm.
- Learning to juggle.
- Learning to volley a simulated ball into a simulated bucket.

Before the conclusion there is discussion of two additional investigations relating this work to other work: a new method of implementing Albus' CMAC and some experiments with "variable resolution dynamic programming".

Chapter 2

What are Robotic Tasks?

This chapter serves as a simple introduction to some of the issues of robotic control. It begins by introducing and giving examples from the disciplines of (i) robot modelling, (ii) robot control and (iii) robot intelligence. It then discusses the problems of conventional robot modelling and how they affect the higher levels of control.

2.1 Conventional Robot Control

This section provides a brief introduction to the tasks facing the designer of a robot controller. I begin by listing the problems which need to be solved in increasing level of abstractness.

2.1.1 Robot Modelling

Conventionally, modelling is achieved analytically. This is a successful and almost universally applied method in many branches of engineering. A set of primitive axioms which model the behaviour of the primitive components of the physical world are combined using mathematical analysis to model complex systems.

Perception. In order to achieve a task, it is often necessary for objects in the real world to be observed, and from these observations to obtain their real world positions and orientations. An example method of perception is vision, in which the mapping from the image to the real world position and orientation is required.

Kinematics. It is usually necessary for a robot controller to obtain the positions and orientations of particular links and joints in different frames of reference (such as the real world). This computation takes as input (i) the fixed data about the robot, such as its topology and link-lengths, and (ii) a vector $\mathbf{q}$ of current joint positions. A joint position is typically either a *joint angle* if the joint is revolute (as are the joints in Figure 2.1) or else a *joint length* if the joint is prismatic. The output of the computation is the location of the links in world coordinates. Conversely, it is often necessary to take as input a target position in some other coordinate frame (such as the real

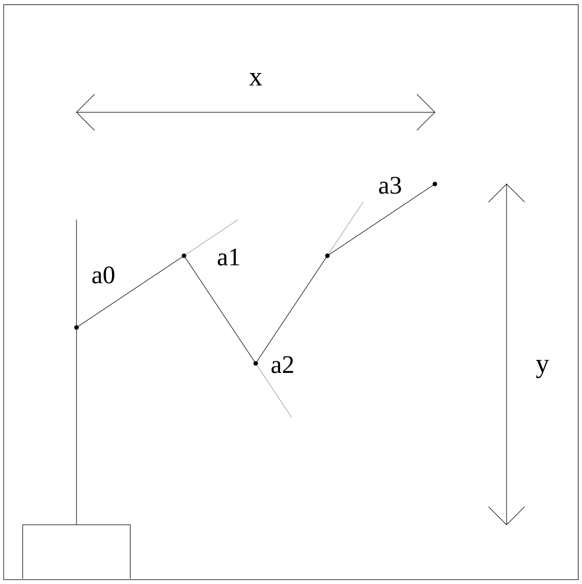

Figure 2.1

A multi-jointed robot manipulator.

world), and to obtain a set of joint positions which would produce this target position. The former computation is of the robot's *kinematics* and the latter is of the robot's *inverse kinematics.*

Dynamics. The robot's dynamic behaviour is determined by the forces which are acting upon it. Some of these forces, such as gravity, are out of its control. Other forces (or torques) are supplied to the robot's joints. The dynamics problem is to compute how the behaviour is affected by the forces acting on the robot.

To formalize this problem, we use the notion of a system's *state.* A *state* representation is a collection of values which contains sufficient information to predict, in principle, the future behaviour of the system, provided the future external and internal forces are also known. A particularly convenient state representation for a robotic manipulator consists of two vectors $\mathbf{q}$ and $\dot{\mathbf{q}}$ which represent the current set of joint positions and their velocities.

The state change, the time derivative of the current state, is determined by the internal and external forces on the arm. Given a current state $\mathbf{s} = (\mathbf{q}, \dot{\mathbf{q}})$, the time derivative of the position component of the state can be calculated from $\mathbf{s}$ trivially—it is the velocity component $\dot{\mathbf{q}}$. The derivative of the velocity, the *joint accelerations* vector $\ddot{\mathbf{q}}$, is the behaviour, dependent on $\mathbf{q}$, $\dot{\mathbf{q}}$ and the *joint torques* $\boldsymbol{\tau}$. This calculation is the dynamics problem. The inverse dynamics problem is the converse. Given a current state $(\mathbf{q}, \dot{\mathbf{q}})$ and a target joint acceleration $\ddot{\mathbf{q}}$, one must compute a set of joint torques, $\boldsymbol{\tau}$, to achieve the target.

2.1.2 Robot Control

Trajectory Tracking. A *trajectory* is a temporal sequence of states

$$((\mathbf{q}_0, \dot{\mathbf{q}}_0), (\mathbf{q}_1, \dot{\mathbf{q}}_1), (\mathbf{q}_2, \dot{\mathbf{q}}_2), \ldots) \tag{2.1}$$

It can be tracked by a sequence of joint accelerations $(\ddot{\mathbf{q}}_0, \ddot{\mathbf{q}}_1, \ddot{\mathbf{q}}_2, \ldots)$ where

$$\ddot{\mathbf{q}}_i = \frac{\dot{\mathbf{q}}_{i+1} - \dot{\mathbf{q}}_i}{h} \tag{2.2}$$

where h is the time step, typically between 1/50th and 1/1000th of a second. $1/h$ is known as the *sampling frequency*. The inverse dynamics model can be used to determine a sequence of joint torque vectors $(\boldsymbol{\tau}_0, \boldsymbol{\tau}_1, \boldsymbol{\tau}_2 \ldots)$ which would cause these ideal accelerations. This method implements open-loop control, and as a result the sequence of torque vectors can be precomputed prior to trajectory execution. For closed-loop control the current state is monitored, and the actual torque vector is modified according to the actual current state.

The advantage of closed-loop control is that, should the behaviour of the manipulator differ from that predicted by the dynamic model, the tracking error can be compensated. There are a variety of reasons that the predicted behaviour is likely to be inaccurate, and these are discussed in Section 2.2.

The compensation can be a function of the error signal. The field of *Control Theory* [Burghes and Graham, 1980] provides a selection of schemes for generating this modification, and also provides the mathematical tools to analyse the stability of the modification strategy. Three common examples are

- **Proportional** (positional) control which adds to the basic precomputed torque a component which is proportional to the current position error, tending to cancel it out.
- **Derivative** (velocity) control which adds a component proportional to the current velocity error. For example, if the required state is stationary, then torques are supplied in the opposite direction to current movement.
- **Integral** control in which the modification varies according to the recent local accumulation of errors.

Different controllers can be combined additively. An example is PD-control in which the chosen torque, $\boldsymbol{\tau}_{\text{now}}$, is defined as

$$\boldsymbol{\tau}_{\text{now}} = \boldsymbol{\tau}_i + K_p(\mathbf{q}_i - \mathbf{q}_{\text{now}}) + K_v(\dot{\mathbf{q}}_i - \dot{\mathbf{q}}_{\text{now}}) \tag{2.3}$$

Such a use of a precomputed inverse model and feedback control is called *feedforward* control. It requires the inverse dynamics model to precompute the necessary torques, as well as general control-theoretic mathematical tools. In particular, the feedback matrices such as K_p and K_v in Equation 2.3 (called *gains*) can be determined analytically. They are chosen to provide a stable response in a short time. Generally, this mathematical analysis requires the assumption of local linearity in the dynamic model.

Because the error is monitored and reduced, acceptable performance can occur even if the model is only a simple approximation. The extreme case is where no inverse dynamics are computed, and each joint torque is computed entirely according to the current position and velocity errors. Each joint actuator is an independent *servomechanism*: a one-variable control system which continually tries to track the input signal with its output signal by means of linear feedback control. For speeds which are not low, this extreme approach results in large trajectory tracking errors.

Another scheme is *Computed Torque Control* [Fu *et al.*, 1987; An *et al.*, 1988], in which modification can also be based directly on the inverse dynamics model. The ideal acceleration to take us back to the trajectory is computed, and then the torques to achieve this acceleration are computed using the inverse dynamics.

If $(\mathbf{q}_{\mathrm{now}}, \dot{\mathbf{q}}_{\mathrm{now}})$ is the current state, and we are meant to be at the ith state in the trajectory, then we attempt to apply acceleration

$$\ddot{\mathbf{q}}_{\mathrm{now}} = \ddot{\mathbf{q}}_i + K_p(\mathbf{q}_i - \mathbf{q}_{\mathrm{now}}) + K_v(\dot{\mathbf{q}}_i - \dot{\mathbf{q}}_{\mathrm{now}}) \tag{2.4}$$

If the trajectory is controlled in this fashion by accelerations it is a *linear system*, defined in Section 4.2. It can be shown that, provided the gains are not too high, this will converge to an accurate tracking of the trajectory.

Computed torque control is computationally expensive because the inverse dynamics must be computed in real time. The extent to which the model is correct affects the accuracy and stability of the trajectory tracking. At some cost in accuracy a simpler model of the robot dynamics could be used. An example would be a model which only took gravitational forces into account.

In summary, trajectory tracking can be achieved by suitable use of the inverse dynamics model. Three possible methods are

- Open-loop control, using precomputed inverse dynamics. This does not check for errors.
- Closed-loop control using precomputed inverse dynamics and also modifications as a function of the error signal. This requires extra mathematical models and analysis.
- Closed-loop control by dynamically computing the ideal current accelerations and in turn the torques to achieve the accelerations.

Balancing. Typically, remaining in a static position is an easy application of closed loop control, where the error signal is simply the difference between the current state $(\mathbf{q}, \dot{\mathbf{q}})$ and the ideal state $(\mathbf{q}_{\mathrm{ideal}}, \mathbf{0})$. The state is always changed to lessen the error. However, in some dynamic situations, no error-decreasing joint torques may be available. This can occur if any of the joint actuators are insufficiently strong to provide the required torque. In some systems there might even not be an actuator at the joint. In this case, a global strategy for returning to the goal is required: local adjustments are insufficient. This is a stability problem. Perfect knowledge of the kinematics and dynamics would not be sufficient to solve this directly; some intelligence is also needed to develop a strategy for stability.

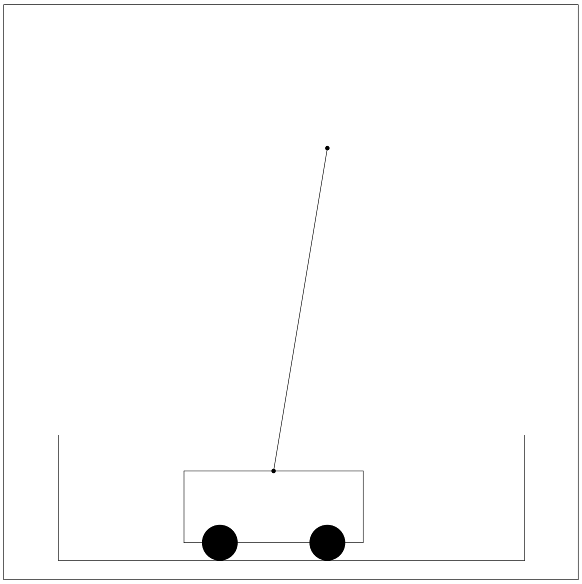

Figure 2.2

The Cart-Pole problem.

The classic example is the pole balancing problem, described in [Michie and Chambers, 1968]. This is depicted in Figure 2.2. The cart can be moved left or right along a bounded track. The base of the pole is fixed to the cart by a revolute joint with no actuator. The only control over the angle of the pole is thus indirect, by means of thrusts to the cart. Balance has to be non-local: there are states in which to prevent eventual disaster, the state vector must be moved further from the goal state.

2.1.3 Robot Intelligence

Obstacle Avoidance. To move the manipulator from one static configuration to another, the controller can track a straight line, uniform speed trajectory through joint space. However, during the transition the arm sweeps out a volume of space which must not coincide with any obstacles. Obstacle avoidance involves finding a trajectory in which none of the intermediate configurations cause a collision.

For obstacle avoidance, a specification of the shape of the robot is required, which is defined as a mapping from $\mathbf{q}$, the joint angles, to the space of three-dimensional solids. This *spatial model* is implemented by combining standard three-dimensional geometrical algorithms with the robot kinematics. A candidate trajectory can be tested by using the spatial model to ensure that no $\mathbf{q}_i$ in the trajectory maps to a solid which intersects with any obstacle. One possible method of solution is to obtain trajectories using a generate-and-test procedure. The search for a valid trajectory can be guided by an evaluation function. This function scores different configurations badly for points near obstacles and well for points far away. The trajectory search follows directions down the gradient of this function.

Autonomy. A highly sophisticated robot controller would require abilities which are at the moment only available to biological systems. These include planning, inter-agent communication and failure management. Such abilities would require a knowledge of the environment which might have to be obtained by learning. The study of these problems are not restricted to robotics, but form much of the general field of Artificial Intelligence.

For example, [Firschein and others, 1986] considers the design of an autonomous robot to assist in the building of a space station. The bandwidth for communication with the robot would be low, unreliable, and with a time delay. As a result, it would not be practical to have a human teleoperating each task, and yet a fixed program specification would not cover the details of the wide range of tasks which the robot would be expected to achieve. Instead, the robot would be given relatively abstract task specifications, and would be left to compute locally the means to achieve them.

2.1.4 Discussion

These were examples of the tasks facing robotic designers. Others include Actuator Modelling, Control Languages and Trajectory Planning and Optimization. The examples were in a roughly increasing order of complexity, and generally the earlier tasks can be used within the solution of later tasks.

2.2 Robot Control: Difficulties

The previous section considered some of the issues addressed by the designer of a robot control system. Now let us explore the difficulties with these conventional solutions. The discussion is split into two sections. The first discusses the problems of robotic mathematical modelling and the second considers the effect that deficiencies in the models have on robot control.

2.2.1 Mathematical Modelling of the Robot's World—the Problems

1. The mathematical model is built from a set of axioms of the physical behaviour of the world (such as Newton's laws of motion). Some low level system components are treated as being atomic, having the ideal properties required to fit these axioms. As a result, the extent to which the model is correct depends on the extent to which the low level components do in fact match the requirements of the axioms. For example, the joints in the arm may be modelled as two rigid links rotating around a common frictionless axis. If in reality there is friction in the joint then this mismatch between the idealized assumption and the world can lead to general model inaccuracies. If instead the friction is modelled, it might be treated as growing linearly with angular velocity. This too, is only an approximation, and the designer must check experimentally that the inaccuracies are sufficiently small.

2. The mathematical model describes the behaviour in terms of the explicit system variables such as angles and velocities and also a large number of system parameters. The system parameters include features like the mass and moments of inertia of each link, link lengths, coefficients of friction, sensor locations, and the relationship between actuator signals and the torques produced. The values of these parameters cannot be obtained from the theory, and so must be measured. Many of these measurements cannot be made directly and so have to be computed from observations of other features of the system (these computations will in turn be based on mathematical models, with the same set of problems). The accuracy of these parameters affect the model accuracy, often critically.

3. The short term dynamic changes in the system's state are included in the dynamic model. However, it is likely that there will be other changes in the system with time. These include gradual changes in the performance of components due to wear and tear (for example, the joint becomes less stiff). Gradual changes could theoretically be incorporated, but in practice there are no techniques to model such wear. Other changes are the result of unpredictable system perturbations (for example a camera being jogged). These, again, are not possible to model. The system designer must assume that small changes have little impact on the model accuracy, and must regularly inspect the equipment to ensure that no significant changes have occurred.

4. Generally, the differential equations resulting from the models are not analytically soluble and so need to be approximated, or computed numerically, in which case issues of numerical stability arise.

5. The generation of mathematical models evidently requires a great deal of expertise. The designer needs to apply knowledge about the axioms of the physical world behaviour. It is necessary to judge which aspects are important to model and which are unlikely to affect accuracy. Then a large degree of mathematical dexterity is required to combine the components of the model and solve the resulting equations. Some aspects of this system might be automatable using algebraic manipulation software such as REDUCE [Hearn, 1973], but generally it requires the time of expensive, highly skilled experts.

6. The mathematical models require enormous computational power. This greatly increases the expense of calculating real time dynamics and inverse dynamics for tasks such as trajectory tracking. However, increasingly powerful computers will eventually be able to deal with this problem even for complex dynamic systems.

2.2.2 Discussion of Robot Control

The argument in this section will be that the high level robotic problems are always one of the following:

- **Search problems.** These are examples of more general problems, particularly those from the fields of optimization and Artificial Intelligence (AI).

- **Modelling problems.** These are caused by inaccuracies in mathematical modelling.

This argument is supported by considering some of the tasks discussed in the previous section. Trajectory tracking is very poor if it is open-loop and the model is even marginally inaccurate. If control is closed-loop then the error caused by an inaccurate model is reduced, and typically consists of the actual state lagging behind, or never quite reaching, the desired trajectory. A further problem for computed torque or feedforward control for trajectory tracking is the enormous amount of computation required. However, other than these modelling problems, there are few difficulties in designing a trajectory tracker.

Balancing can be achieved by trying to track the static trajectory. It is made harder in the case where the system is in a state of serious imbalance: an attainable trajectory back to safety must be deduced, and there is no entirely general way to obtain such a trajectory analytically from the mathematical model. If the strategy is to be obtained automatically, search techniques must be applied.

Obstacle avoidance relies on a spatial kinematic model. Given a goal and a set of known obstacles then even if the model is entirely correct, it will not be able to prescribe a suitable trajectory. The solution is a search problem in three-dimensional space.

Autonomy depends on an abstract representation of the world and upon many of the general problem solving abilities proposed and investigated in a variety of fields in AI. Abstracting some aspects of the world, such as placing everything in a uniform coordinate system, is a modelling problem. Other aspects of autonomy are from the AI domain.

This dissertation concentrates on a solution to modelling problems. When the designer of the high level control is confronted with a search problem they have two alternatives

1. To perform the search and planning tasks manually, and encapsulate the knowledge in a program.

2. To use the best automatic techniques from the optimization and AI fields.

The conclusion is that for many robotic design tasks, at least one of these alternatives will be entirely adequate. This is because the design tasks may be in domains which are restricted enough to allow such knowledge to be expressed, or for planning to take place in realistic time. The majority of the dissertation will focus instead on how to reliably and accurately learn models of the world.

Chapter 3

Learning Robotic Tasks

This chapter surveys the related literature in learning control. It begins with early work, and then introduces the important distinction between work which learns action maps and work which learns world models. A particularly important aim of the survey is to show that earlier work in the field can usually be regarded as learning some relationship between state, action and behaviour. Other important issues in learning control are also surveyed. Following that, some particularly relevant recent work is summarized. At the end of the chapter is a description of how this dissertation fits in with the issues discussed.

3.1 The Birth of Learning Control

The mechanical governor, invented by James Watt in 1788, is a device which regulates the speed of a rotating shaft. If the rotation increases then a pair of balls move further apart. This movement is transmitted mechanically to the engine producing the rotation, and causes it to work less hard, for example by closing a gas flue. Conversely, if the rotation is less than ideal, the balls are closer and this similarly causes the engine to increase its output. It is an elegant system which controls itself automatically, by sensing the performance and then making adjustments to bring the performance closer to ideal. Such systems have been common in mechanical engineering for over a century. These are called *closed-loop* control systems.

During this century, the use of analogue electronics has allowed a much more flexible approach to such systems. Measurements of the performance can be encoded as electrical signals instead of mechanical positions. From this, more sophisticated forms of closed-loop control have been possible. More recently, the signals have been encoded and processed digitally, allowing yet another increase in the flexibility and amount of the information available.

The large amount of information which can now be obtained about a system leads to the question of how it can best be used. One approach is to use it in the same manner as earlier generations of controllers. This provides robust but mundane control. An example of this approach are servo-driven robotic manipulators. Each joint is moved by specifying the desired position of the joint, and then this simple closed-loop control is used for each joint independently to continually make adjustments until the position of each joint is as required.

A more interesting possibility is to use more of the information received through the sensors to autonomously improve performance by *learning* about the world. The benefits are summarized here.

- **Optimality.** Control of complex systems such as manipulators can become quicker and more accurate.
- **Self Programming.** Complex systems do not need to be analysed and modelled by expensive human experts. Instead they can discover their own abilities.
- **Model of Intelligence.** A system which improves its own performance provides a possible model for the behaviour of biological systems.
- **Disorder.** A system which monitors its own performance might be able to cope with a noisy, or non-stationary environment.

The difficulty is that there is a very large amount of information which can be used. Many trade-offs are available between (i) the reliance on standard control techniques and (ii) the intensive processing of dynamic information. One trade-off is to use the information to automatically make adjustments to the controller at a high level. This is discussed in the next subsection.

3.1.1 Adapting the Higher Levels of Controllers

The standard closed-loop controller makes adjustments at a low level. If a servo-controlled arm tries to follow a trajectory it continually monitors the current error in the position and velocity. In addition to this, adjustments can be made at a higher level of control. Thus, after an attempted trajectory has been completed, the data collected can be used to adjust, in advance, the torques which will be supplied to each joint at each time step next time the trajectory is attempted. This idea is the basis of **Adaptive Control** [Phan *et al.*, 1990].

A different example of the idea of making adjustments at higher levels of abstraction is **Task-Level Learning** [Aboaf *et al.*, 1989]. An example of this is throwing a ball at a target. Conventional modelling and control are used to throw the ball, but when it fails (due to modelling errors) an adjustment is made to how hard the ball is thrown. If it had fallen short it is thrown harder next time, and if it had overshot it is thrown less hard. The idea was applied successfully to a real bat and ball juggler, the lower levels of which had been built using mathematical models of the system components. With no task-level learning the ball was usually dropped after only two or three hits. With task level learning the time to failure was substantially increased to typically between eight and twenty hits.

3.1.2 The Basis of Learning Control

The basis for learning control was established during the 1960s and is consolidated in the expository paper [Fu, 1970]. Here, I summarize the paper. Fu cites a variety of possibilities for learning control.

- **Classification from a teacher.** The first possibility was to learn to *classify* states according to which action should best be applied. This drew on the emerging field of **Pattern Classification** [Duda and Hart, 1973]. If a sample of states with known optimal actions are given, then a pattern classifier can be learned which, when given a state not in the original sample, produces an action which, according to some predefined criterion, best agrees with the sample points. Possible criteria included the assumption that the optimal control actions were *linearly separable*, in which case the learning controller tried to find a hyperplane to partition the state space in a manner which agreed with the sample points. This method did not actually use information from the environment to improve performance, but instead relied on a teacher to tell it what was correct.

- **Reinforcement learning.** If, after each action is applied, part of the information from the environment is a signal saying how ideal that action had been, then this signal could be used to improve performance. This is *reinforcement learning.* In this work the state space was partitioned into different regions. Within each region the relative probabilities of attempting alternative actions were modified according to the reinforcement signal. The use of a reinforcement signal increases the autonomy of the learning, but is still a tricky requirement: many systems, while having a global goal, find it hard specify local goals which would lead to the global performance.

- **Stochastic automata.** A stochastic automaton has a finite number of internal states which are traversed between each control cycle. The input to the automaton is one of a finite number of reinforcement signals from the world. The output of the automaton is one of a finite number of actions it applies. The state transition function is a stochastic matrix, which is modified by a scheme designed to increase the probability of causing outputs which produce higher levels of reinforcement.

Fu's overview also mentions some of the issues which, since the date of the paper, have been further developed in the field. These include the problems of learning in a world which can change unpredictably (a *non-stationary* environment), the problems of a slow learning rate and the question of how learning controllers can be linked together. The history of these and other issues are discussed in Section 3.3, but first I will explain how different approaches to learning control developed from these seeds.

3.2 What Should be Learned?

There are two fundamentally different things that a learning control system can attempt to acquire. One is the **Action Map**

$$\textbf{State} \rightarrow \textbf{Action} \tag{3.1}$$

and the other is the **World Model**, which in this disssertation is interpreted as being of the form

$$\textbf{State} \times \textbf{Action} \rightarrow \textbf{Behaviour} \tag{3.2}$$

which is often learned instead as the **Inverse World Model**

$$\textbf{State} \times \textbf{Behaviour} \rightarrow \textbf{Action} \tag{3.3}$$

The inverse form of the world model is dangerous to learn as it may not be a function. Given a current state and a desired behaviour, there might in some contexts be no actions to achieve the behaviour, and in other contexts multiple actions. Successful learning of an inverse world model requires extra domain knowledge that these problems will not occur.

During the remainder of this chapter, all the work will be surveyed using my interpretation of learned mappings as being either action maps or world models that relate state, action and behaviour. The next two subsections describe early examples of each of these. Following that, subsection 3.2.3 contrasts the two approaches.

3.2.1 Michie and Chambers: BOXES

An early, and classic, example of action map learning is the BOXES pole-balancing system [Michie and Chambers, 1968]. The system to be controlled is a pole balanced on a cart. The cart can be thrust left or right at each control cycle, but these are the only permitted actions. The state of the system (which consists of four values: the position and velocity of the cart, and the angle and angular velocity of the pole) can be observed at each control cycle. The goal is to prevent the pole from falling, and this is the only specification which is given to the learning system.

The systems learns the action map:

$$\underbrace{X_{\text{cart}} \times \dot{X}_{\text{cart}} \times \theta_{\text{cart}} \times \dot{\theta}_{\text{cart}}}_{\textbf{State}} \rightarrow \underbrace{\{\texttt{Left}, \texttt{Right}\}}_{\textbf{Action}} \tag{3.4}$$

The representation of the mapping is a four-dimensional array, indexed on the quantized state variables. For example, all x coordinates of the cart between 35ins and 21ins to the left of the cart are considered behaviourally equivalent. Five grades of x position are distinguished. In total there are 225 entries in the array (these entries are the boxes after which the system is named). The contents of the boxes affect what action is chosen should the system's state ever coincide with the box. It contains statistics of previous decisions made when the box was entered. These identify the mean survival time when the `Left` action was taken and mean survival time when the `Right` action was taken. The decision as to which direction to choose is biased according to which is expected to have longer life. Thus as learning continues, decisions which lead to disaster are gradually eliminated, even if they do not lead to immediate disaster.

This system thus displays the desirable property that a local reinforcement signal is not required—all learning is done simply from the final outcome of the trial. The system did indeed manage to learn to balance the pole within typically a thousand balancing attempts.

3.2.2 Raibert's Parameterized Method

Raibert's work [Raibert, 1978a; Raibert, 1978b] learned to control a real torque driven robot manipulator. It learned the following inverse world model:

$$\underbrace{\text{Joint Angles} \times \text{Joint Velocities}}_{\textbf{State}} \times \underbrace{\text{Joint Acc'ns}}_{\textbf{Behaviour}} \rightarrow \underbrace{\text{Joint Torques}}_{\textbf{Action}} \tag{3.5}$$

This is again represented by a quantized multi-dimensional array. The state space is six-dimensional with ten quantization levels and so there are 10^6 cells (hash coded to save memory). Each cell corresponds to a simple local model of the behaviour of the robot. This learning system uses domain knowledge about the form of equations of motion of a robot arm. Such equations have parameters which vary throughout the state space but can be assumed constant within each box. The values of these parameters can be estimated by, for each cell, recording the real world experiences of the robot and then inverting the known (linear) form of the local model to obtain the parameters. The method of estimating local parameters provides a very accurate model with the disadvantage of needing to assume a certain form for the dynamic equations of motion.

The learning system was given the task of following a prespecified trajectory. This it did by using the learned model to precompute the necessary torques. During execution of the trajectory, no feedback was used. Despite this open-loop control, the performance was good and improved during learning (though it did not reach perfect behaviour). Convergence took approximately 2000 trials.

Raibert carried out some further tests illustrating important features of learning control. Firstly he tested performance in an environment which changed unpredictably over time. The experiment consisted of a period of normal learning after which, unknown to the controller, the dynamic behaviour of the arm was changed by adding a weight to a joint. A second experiment similarly attached a spring to a joint. The arm adapted, but slowly, to the changes.

His second extra experiment was to learn one trajectory and then try executing a nearby trajectory. This was to see how successful the *generalizing* abilities of the learning system were. The results showed that a trajectory was learned more quickly if a nearby trajectory had been previously learned.

3.2.3 Learning Action Maps or World Models?

Subsequent investigations in learning control have differed as to which of these two approaches is adopted. This survey will distinguish clearly for each piece of work it examines which kind of mapping is learned. This is because it has a large impact on the applicability, expected performance and utility of the learning system concerned. There is a trade-off between the usefulness of what is learned and the expected ease and speed of learning.

Learning action maps provides a more useful end-product because the learned controller knows what to do at each state it is in. The disadvantage is that an action map is generally hard to learn. It is not always clear whether current performance can be improved, and if it can be, how to improve it. A very simple example of this is the two armed bandit problem, described more fully later, which is a system with no state and only two possible actions but for which an optimal solution is still not known.

Despite these serious difficulties, action map learning has been attempted with some success [Michie and Chambers, 1968; Barto *et al.*, 1983; Kaelbling, 1990b; Simons *et al.*, 1982; Gordon and Grefenstette, 1990]. As well as Michie and Chamber's Pole Balancer, [Barto *et al.*, 1983] have implemented a learning controller which balances a pole considerably more quickly with the same delayed reinforcement signal (remember, the pole controller only gets told about its performance when the pole falls). The improvement is by *learning* an immediate reinforcement signal. The reinforcement signal scores an action decision as good if it moves the pole state into a superior state and bad if it moves it into an inferior state. The relative qualities of states are estimated by a record of the expected time to failure starting from each state. The precise definition of this value is hard to pin down. It is trying to estimate the expected time to failure from the current state if the optimal controller were used, but it is estimating this by means of the expected time to failure if the current controller is used. Unfortunately this recursive definition could have multiple solutions and so is not necessarily well-defined. In practice in this and other work (e.g. [Jordan and Jacobs, 1990]), this ambiguity does not seem to cause a problem.

A very recent investigation [Kaelbling, 1990a; Kaelbling, 1990b] thoroughly consolidates work in this area as "Learning in Embedded Systems".

Learning world models is much easier, because it is based on objective observations about the world. If performance is inadequate then it is because the observed behaviour differs from the predicted behaviour. In such a case it is clear how the world model should be updated—it should reduce or eliminate the error. For this reason, model-based learning control systems have been more popular [Raibert, 1978b; Miller, 1989; Mel, 1989; Atkeson, 1989; Zrimec and Mowforth, 1990; Sutton, 1990]. There is a sacrifice to be made for the relative ease of learning: although the world may be modelled, it is not necessarily clear how to use this model. The investigations mentioned above deal with this problem in a variety of ways.

- **Weak AI or optimization.** In [Christiansen *et al.*, 1990] a controller learns how a flat block behaves when the tray upon which it is lying is tilted by a robot. The experiments involve a real, visually observed, robot. The world model learned is

$$\underbrace{\text{Start pos'n and orientation}}_{\textbf{State}} \times \underbrace{\text{Tilt angles}}_{\textbf{Action}} \rightarrow \underbrace{\text{End pos'n and orientation}}_{\textbf{Behaviour}} \tag{3.6}$$

The representation is by means of a quantized array. The robot is given a goal position and orientation. The current position and orientation is observed. There is generally not an action which could immediately produce the desired goal, and so instead a standard search is carried out with reference to the learned world model to find a sequence of actions to achieve the goal.

The tray tilting work explores further learning control issues discussed shortly.

Other examples of model-based learning which use search and optimzation are [Mel, 1989] which performs a best first search to produce an obstacle-avoiding positional trajectory to reach visual goals and [Sutton, 1990] which uses dynamic programming based on the learned model to plan simple maze paths to a goal.

- **Perform a non-abstract task.** Some robotic tasks are sufficiently concrete that there is not much more to do than learn the world model. The prime example of this is the trajectory tracking task studied by Raibert, and similar tasks are in [Atkeson, 1989; Atkeson and Reinkensmeyer, 1989; Miller *et al.*, 1987].

- **Use a model-based pre-programmed controller.** This is a logical extension of the previous approach. Given an abstract problem, design a model-based controller to achieve the problem. Such model-based controllers can be simple and unsophisticated. A recent example of this is the controller for a visual tracker designed in [Miller, 1989], which is guided by a program that can be expressed as a short set of decisions and feedback rules. The top-level programs are of a sufficiently simple form that there may be some mechanism to generate them automatically.

- **Learn an evaluation function.** As well as learning the world model, an evaluation function on world states can be learned. An *evaluation function* is a mapping of the form

$$\text{EF} : \textbf{State} \rightarrow \Re \tag{3.7}$$

Conventionally, the lower $\text{EF}(s)$, the better the state s. The world model in conjunction with the evaluation function can be used to choose actions. Given a current state, the set of possible actions is consulted, and for each candidate action the evaluation is computed of the predicted resultant state. The action is chosen which minimizes the predicted evaluation. This is efficient provided the number of possible actions is not large.

This method is used by [Connell and Utgoff, 1987] to balance a pole, though in this case the world model is not learned, but estimated from the single previous state transition. A mapping is learned from states to the expected time to disaster. Learning evaluation functions has also been used in other domains such as puzzle and game learning, where the world model is trivially available and does not need to be learned [Samuel, 1967; Rendell, 1983].

3.3 The Important Issues for Learning Control

3.3.1 The Curse of Dimensionality

Realistic systems, whether they are learning action maps, or models of the world, should be able to cope with domain dimensions between approximately zero and eighteen (eighteen, because a direct drive six-jointed arm has twelve dimensions to its state space and six dimensions to its action space). However, much work has been restricted to learning task dimensions between zero and four. As we will see, many learning representations and convergence times become exponentially worse with increasing dimensionality. Approaches which denumerate all possible actions similarly blow up with increasing dimensionality of the action space.

These problems are compounded when, as is usually the case, the variables of state space and action space are continuous. For example, it is generally not known in advance at which level it is safe to quantize, or whether the quantization levels should vary.

The problem of dimensionality has rarely been directly addressed in the learning control literature. It is generally dealt with in one of the following ways.

- **Assume the control spaces are small and denumerable.** This is the assumption of stochastic automata [Fu, 1970], and of systems which make brutal quantizations to state spaces [Michie and Chambers, 1968; Barto *et al.*, 1983; Christiansen *et al.*, 1990], and commonly for reinforcement learning research [Kaelbling, 1990b; Sutton, 1990]. Similarly action spaces are often small (for example the classic pole balancer has only two actions). This is a reasonable approach for initial investigations of other aspects of learning control, but there is no doubt that it is useful, at some point, to take these initial approaches up to bigger problems.

- **Assume there is underlying, discoverable, structure in the problem.** To generalize in any way it is essential to have this assumption in some form. However, the strength of the assumption can vary very greatly. Parameterized mapping learners, which are described in Section 5.2, and which include polynomials and neural nets, use a strong form of the assumption [Minsky and Papert, 1969; Jordan and Jacobs, 1990]. Unless chosen with foreknowledge of the structure of the world model, there is no guarantee that any possible set of parameters could produce a mapping which would adequately model the data. Decision tree classifiers [Quinlan, 1983] make a weaker assumption—that the domain can be split up into a fairly small number of large hyperrectangular regions in which the classification is constant. This feature is common with the approach of [Salzberg, 1988] and [Aha *et al.*, 1990], which both learn using the nearest neighbour generalization, but which use the assumption that classification regions can be characterized by a small number of well chosen example points (*exemplars*).

- **Only learn about one task.** Even if the state space is eight-dimensional, if only one trajectory is required, then the behaviour of the world need only be learned along one one-dimensional strand. This is the approach used by adaptive controllers for robot arms [Phan *et al.*, 1990; An *et al.*, 1988]. It was also used in [Miller *et al.*, 1987]. The dimensionality of the model is thus brought down to one.

- **Only learn small sub-areas of the task.** This is a natural extension of the previous approach, which learns small regions of the domain, but not as small as a one-dimensional strand. Even for an entirely repetitive task it is usually important to know about behaviour which is close to the solution of a task, but which is not actually in the solution of the task. This is in order to compensate for unpredictable deviations. Thus the controller's tactics in learning a task are to try to keep the experiences clustered around a fairly low dimensional, task-specific, subspace. In [Miller, 1989] this goal is stated. This idea is demonstrated in [Clocksin and Moore, 1989], in which a hand-eye coordination relation is learned for a five-jointed arm, but learning is biased to explore a two-dimensional subspace sufficient to reach all observed positions.

3.3.2 Variable Resolution

The ability to concentrate on particularly important areas of the control space requires a suitable choice of mapping representation. This aim is particularly important given the conclusion of the previous subsection—that the only defence against the curse of dimensionality without assuming extra domain knowledge is to concentrate on task-specific sub-areas.

This issue is considered in [Simons *et al.*, 1982] in which array boxes are recursively partitioned to increase resolution where necessary and also in the work of [Connell and Utgoff, 1987] which learned to balance a pole without needing to quantize the state variables.

3.3.3 Modularization

In almost all technological professions, large systems are broken down into smaller components—typically in a hierarchy. It is desirable to achieve this with learning control systems. The result would be a group of simultaneous learning controllers, with some abstract controllers making use of other concrete controllers. There are two issues here:

- How should the hierarchy be organized?.
- How could the organization be achieved automatically?

The first question has been mentioned in several places [Fu, 1970; Sutton, 1990], but has not been discussed in detail. The exception to this is the architecture proposed in [Albus, 1981]. The second problem is interesting but very difficult, and has not been addressed for hierarchical structures with modules as complex as learning controllers.

3.3.4 Disorder in the Environment

The consequence of a disordered environment is that individual observations may not be reliable. The following are reasons that an environment may be disordered.

- **Noisy environment.** What the controller perceives is randomly perturbed from what actually happens. In this case the solution is to use local averaging of data, the implementation of which depends entirely on how the mapping is represented. Representations of mappings are discussed in Chapter 5.
- **Non-deterministic environment.** A simple approach to this problem would be to treat the non-determinism as noise. For interesting forms of non-determinism this is inadequate because the variation and probability distribution of the mapping being learned can also be valuable information for the controller. The tray tilting robot of [Christiansen *et al.*, 1990] learns to use actions which have minimal non-determinism in preference to unreliable actions. Non-determinism in the learning of action maps is also considered by [Kaelbling, 1990b; Sutton, 1990].
- **Non-stationary environment.** This problem has been investigated by [Raibert, 1978b; Miller *et al.*, 1987; Moore, 1990] for the case where the world which is being modelled is perturbed, requiring a quantitative change in the control strategy. A much harder problem is discussed in [Sutton, 1990], in which the control strategy can undergo a qualitative change. Sutton's DYNA-Q system is described in Section 3.4.

3.3.5 State Identification

Both action maps and world models need to be able to detect the state of the system. At this point it is worth recalling the definition of a system's *state*. Imagine that we can detect a certain amount of information about the current configuration of the system. If this information, combined with any proposed sequence of actions is *in principle* sufficient to determine future behaviour of the system, then the information is a representation of the system's state.

Most work in the field assumes the information provided to the system is sufficient to determine state, thus requiring a certain (although small) amount of world knowledge from the system designer. A partial solution to the case where the important aspects of state are unknown is proposed in [Simons *et al.*, 1982; Farmer and Sidorowich, 1988]. Another approach is suggested in [Vogel, 1989].

3.3.6 Experimenting

When the controller is learning, it needs to generate a diversity of experience. Methods to achieve this fall into three categories.

- **Use a teacher.** The role of a teacher is not simply to tell the system what should be done to perform the task, but can also be to guide the system to areas of the state space which are judged to be profitable to explore. The most common form of teacher has been a naive servo (linear feedback) controller [Atkeson and Reinkensmeyer, 1989; Miller, 1989; Miller *et al.*, 1987] which directs the experience towards areas of the state space which lie close to the solution. Extra domain knowledge of the structure of the world model is required to provide such a teacher.

- **Use randomness.** This is the most common approach to gaining experience. Recent examples of its use have been [Mel, 1989; Zrimec and Mowforth, 1990].

- **Estimate the utility of information-gain.** This has been recently investigated thoroughly by [Kaelbling, 1990a; Kaelbling, 1990b]. This work uses a statistical heuristic called Interval Estimation to choose actions which are likely to achieve reward, but which avoid getting stuck on repeated application of a known mundane action when there are superior actions with little experience available. The work investigates (i) algorithms in which there is immediate reinforcement, and (ii) delayed reinforcement (so that the choice of action is not only motivated by the next state, but perhaps by many states in the future). It also copes well with very non-deterministic environments. Choosing actions using heuristics which include the benefits of information gain has also been investigated by [Christiansen *et al.*, 1990; Sutton, 1990].

3.3.7 Inductive Learning

The discussion, and literature reviewed in this section, has been considering the design of a controller to perform well in its environment. There are other goals of learning, and one important one is to *explain* the environment. It can be argued that unless this is achieved, truly complex tasks will always require human intervention. Furthermore, it has been argued by [Michie, 1989; Sammut and Michie, 1989] that learning systems will not be accepted commercially unless their decisions can be understood by the human users.

3.4 Learning Robot Control: Recent Work

3.4.1 Connell and Utgoff: Variable Resolution Pole Balancer

In [Connell and Utgoff, 1987] an evaluation function was learned for the classic pole balancing problem (described in Section 3.2.1). The evaluation function was

$$\underbrace{\text{Cart-Pole state}}_{\textbf{State}} \rightarrow \textit{Desirability} \tag{3.8}$$

As mentioned in Section 3.2.3, an evaluation function in conjunction with a world model can provide the same functionality as an action map, but in this experiment no world model was used. Instead the evaluation of the next state if the most recent action were repeated is obtained. This is obtained using the behaviour of the most recent action in the previous state as a guide to how it would alter the current state. If the predicted evaluation is a decline then the alternative action is automatically used without predicting its consequences.

The evaluation function is represented by an explicit record of experienced states and interpolated using Shepard's method, described in Section 5.2. The evaluation function is not updated using the relationship between subsequent states, but according to an ad-hoc analysis of the 100 states prior to the collapse of the pole. However, with an appropriate choice of parameters it does quickly learn to balance the pole.

3.4.2 Miller: Learning World Models using CMAC

Recent work by W. T. Miller and colleagues [Miller *et al.*, 1987; Miller, 1989] has used CMAC [Albus, 1975a; Albus, 1975b] to model the world. The model is then used in conjunction with a teacher, in the form of a simple linear feedback controller, to improve performance. Two investigations have been reported in the literature.

- **Learning to track dynamic trajectories.** This work learned the inverse world model

$$\underbrace{\text{Joint angles} \times \text{Joint velocities}}_{\textbf{State}} \times \underbrace{\text{Joint Accelerations}}_{\textbf{Behaviour}} \rightarrow \underbrace{\text{Joint Torques}}_{\textbf{Action}} \tag{3.9}$$

  for a simulated two-jointed robot arm. Goal trajectories were defined in joint space coordinates. The main experiments used a repetitive trajectory, and were taught by a fixed-gain controller. The results showed quick improvement on the performance using the feedback controller alone. Experiments were also carried out with several trajectories to be learned and with changes in the environment. CMAC's behaviour was discussed, in particular the problems of too small an underlying memory.

- **Kinematic visual tracking of moving objects.** A three-jointed robot arm held a camera, pointing downwards at a conveyor belt. The work was motivated by the advantages of being able to use world models in the same coordinate system as the task being learned. In this case the task coordinates were the visually sensed position and orientation of a plastic disposable razor and the sensed joint angles of the arm. The arm was controlled by requesting joint velocities which were obtained by independent servo motors in each joint. Both the forward and inverse world models were learned. The forward model was the relationship between the current joint angles, the current observed position of the razor, the requested joint velocities that were sent to the motors and the resulting change in the image position. The image position value was obtained by image processing, and consisted of three values: the x and y coordinates of the center of the razor's image and its orientation. The forward model was thus

$$\underbrace{\text{Joint angles} \times \text{Image position}}_{\textbf{State}} \times \underbrace{\text{Joint velocity}}_{\textbf{Action}} \rightarrow \underbrace{\text{Image change}}_{\textbf{Behaviour}} \tag{3.10}$$

and the inverse model was

$$\underbrace{\text{Joint angles} \times \text{Image position}}_{\textbf{State}} \times \underbrace{\text{Image change}}_{\textbf{Behaviour}} \rightarrow \underbrace{\text{Joint velocity}}_{\textbf{Action}} \tag{3.11}$$

The task was to keep the image of the razor fixed (which meant the arm had to move to keep the camera still relative to the moving razor). Experience was again provided by means of a teacher, a fairly complex position feedback controller. This required a substantial amount of domain knowledge, because the feedback was in joint space, whereas the tracking error was in image space. Two CMACs were used, one for each model. On each control cycle the forward model was used to predict where the razor would appear on the next cycle, and from this the desired image-position change was computed. The backward model was used to obtain a joint velocity to achieve the desired image-position change, and the feedback control signal was added. The results were again good, with a final average error of approximately a quarter of that obtained with the feedback control alone. Learning typically took about ten trials, with the razor being placed identically at the start of each trial. With random initial razor configuration, learning was considerably slower and less accurate.

3.4.3 Mel's MURPHY

This work [Mel, 1989; Mel, 1988] learned vision-based kinematic control of a real three-jointed planar arm. An interesting feature of the investigation was an "ecological" approach, in which the visual observations were kept in the raw sensed form of a 64×64 binary array. The control was again based on a learned world model:

$$\underbrace{\text{Joint angles}}_{\textbf{Action}} \rightarrow \underbrace{\text{Raw Image}}_{\textbf{Behaviour}} \tag{3.12}$$

The world model was forward. As Mel explains, this is generally the only valid direction in which to learn since the inverse model will usually not be a well-definable function. While true of most domains of other workers, this is particularly true of the kinematics of a *redundant* manipulator. A manipulator is redundant if there are multiple ways to move the gripper to a desired position (or desired position and orientation). The world model is learned by a period of random flailing of the arm. It is processed and represented by a kd-tree algorithm which has the behaviour of a neural-net (Chapter 6 introduces, describes and evaluates kd-trees).

After the world model is learned it is used to plan sequences of incremental joint modifications to reach target positions while avoiding visually observed obstacles. Mel stresses the importance of this planning taking place using the learned model rather than requiring real execution. The plan is a modified best first search using a visual distance heuristic. It is aided by a second learned world model—the inverse differential kinematics:

$$\underbrace{\text{Joint angles}}_{\textbf{State}} \times \underbrace{\text{Hand position change}}_{\textbf{Behaviour}} \rightarrow \underbrace{\text{Joint angle change}}_{\textbf{Action}} \tag{3.13}$$

This too is not a well-defined function, but Mel explains how to rectify this. The learning is successful, but computationally expensive (though much of the expense seems likely to be due to processing of the 64×64 images).

3.4.4 Atkeson's Memory-based Control

This work [Atkeson and Reinkensmeyer, 1989; Atkeson, 1989] learns to control a simulated dynamic robot arm to follow a trajectory and also learns corrections to a foot placement model for a simulated hopping robot. For the first experiment the model learned is the inverse world model

$$\underbrace{\text{Joint angles} \times \text{Joint velocities}}_{\textbf{State}} \times \underbrace{\text{Joint Accelerations}}_{\textbf{Behaviour}} \rightarrow \underbrace{\text{Joint Torques}}_{\textbf{Action}} \tag{3.14}$$

The representation is the explicit set of data points. A variety of generalizations are tried:

1. Nearest neighbour.
2. Local regression.
3. Local fitting to a quadratic surface.

The task is specified by a trajectory of joint angles and the experience is gained by means of a teacher—a linear PD-controller. The convergence is generally successful and quick, though the simple nearest neighbour generalization sometimes gets into "stuck states", in which performance is not improved. The likely reason is that an incompletely learned inverse model can resuggest actions which are known to have failed. This problem is discussed in Section 5.1.

The rate of learning and final accuracy is seen to improve with increasing complexity of the method of generalization.

The foot placement task is learned as

$$\underbrace{\text{Hopper state}}_{\textbf{State}} \times \underbrace{\text{Velocity next step}}_{\textbf{Behaviour}} \rightarrow \underbrace{\text{Foot placement}}_{\textbf{Action}} \tag{3.15}$$

Instead of the learning the model directly, it is learned as an adjustment to a simple analytic world model. This difference mapping can be expected to be smoother than the direct model, and thus more easy to learn. Other aspects of the foot control are achieved using non-learning methods. The results show a marked improvement over using the simple world model, though stuck states are still a problem.

3.4.5 Zrimec and Mowforth's Block Pusher

In [Zrimec and Mowforth, 1990] a real robot learns the effects on a block of pushing it with the gripper of a robot. The block's position on a horizontal surface is observed visually, from above, before and after a smooth straight line robot movement. The world model learned is

$$\begin{aligned} &\underbrace{\text{Relative pos'n \& orientation of block}}_{\textbf{State}} \times \underbrace{\text{Relative movement of pusher}}_{\textbf{Action}} \\ &\rightarrow \underbrace{\text{Change in relative pos'n \& orientation}}_{\textbf{Behaviour}} \end{aligned} \tag{3.16}$$

An interesting feature of this experiment is that there is no goal, simply an undirected aim to obtain knowledge. The representation of the mapping is a decision tree, which is able to produce a concise, human comprehensible, description of the mapping. Experimentation is by means of random movements.

In the same investigation the following further issues are discussed:

1. How to decide which variables are dependent on which others.

2. How to quantize the range of continuous variables.

3. Possible methods for automating 1 and 2.

3.4.6 Sutton's DYNA-Q Architecture

This work [Sutton, 1990] has only a rather small experiment, which leads to uncertainty as to whether the approach can scale up. However, there is discussion of a number of interesting and important issues for learning control. The DYNA framework assumes that the specification of tasks is only via reward, or reinforcement, signals. This assumption makes the learning problem more difficult, because, for example it would forbid the use of feedback controllers as teachers. However, if DYNA learning is achieved then the system has a great deal of autonomy.

The system learns a world model, but also generates an evaluation function which estimates the relative qualities of states. The evaluation function is computed from the world model by dynamic programming [Burghes and Graham, 1980]. In practice, to avoid periods of wasted time in which the robot is planning without gaining extra knowledge for the world model, the dynamic programming occurs incrementally, in parallel with task execution.

The example is a simple maze domain with 56 discrete states and 4 discrete actions. The controller receives reinforcement of zero, except when it moves into a special goal state. If it reaches the goal state it is reset to its starting state. Thus, the tactics for maximum overall reward are to repeatedly trace the shortest path from start to goal, but even these abstract tactics are not given to DYNA in advance. Learning, and optimal behaviour, do indeed occur quickly after the first run.

Experimentation is random, but biased in favour of actions which are predicted to produce improved reward. As time progresses, the level of randomness decreases.

To cope with a changing environment, in which the current best set of actions might change, Sutton suggests adding an *exploration bonus* to the reinforcement which encourages rarely visited states to be occasionally visited in case there is possible improvement. This is demonstrated with a maze in which wall sections occasionally appear or disappear.

3.5 This Investigation

Here, I briefly mention which aspects of the field described in this chapter are investigated by this dissertation. I am concerned with making robot learning more practical, and so world model learning is used in preference to action map learning. A very important topic which is tackled is the curse of dimensionality, which has not been a specifically stated goal for other pieces of work. Part of the approach to this is to use a learning method with very variable resolution, and this has profound consequences for the representation of the mapping. The initial representation examined, while ideal for variable resolution, learning rate, and learning efficiency has poor noise tolerance and brittle performance in a non-stationary environment. The method would have a serious weakness could it not cope with these problems and so modifications are developed which do not lose its primary advantages.

A second critical aspect for avoiding dimensionality problems and for increasing the rate of learning is the nature of experimentation, and a method is developed which estimates the utility of information gain.

In order to compensate for the relative weakness of the autonomy of a world model learner (compared with an action map learner), the investigation also focuses on how complex tasks can be modularized into a hierarchy of learning tasks. It also learns as abstract a model as it can by performing entirely in the *perceived* world.

3.5.1 Robustness

Another aim of this work is to have a robust learning method—one in which the system does not get into a situation which does not achieve the goal and in which further improvement does not occur. A learning system which is in such a situation is called *stuck*.

Some identified features which might cause sticking are:

- Insufficiently general class of learnable models. (Discussed for *parametric* methods in Section 5.2, and shown to be avoided in Section 5.3).
- Using only an inverse world model. (Discussed in Section 5.1).
- Failing to provide adequate experimentation. (Discussed in Chapter 8).
- Blame assignment errors in hierarchical learning systems. (Discussed in Chapter 9).

3.5.2 Issues not Addressed

Some important issues are not addressed by this investigation. In each case it is considered that the loss of the feature, while requiring more of the system designer, is also not generally critical for the autonomy of the system. These issues are state identification, learning a non-deterministic world model and providing an inductive ("simple") explanation of the world model. A further issue which is left to the system designer is scaling of state variables (discussed in Section 5.1).

Chapter 4

SAB Learning

This chapter introduces SAB Learning: the main idea which will be developed during the course of this dissertation. The chapter begins with robot problems which have no state, and introduces the concept of a Perception Function. It then extends this to dynamic state and the Perceived State Transition Function. Finally the sequence of actions taken by an SAB learner is described—the SAB Control Cycle.

4.1 AB Learning

Before considering full dynamic control of robots, we will examine the simpler problems of perception and geometry. An example of such a problem is hand-eye coordination.

The end-point of the arm is readily identifiable by some image processing. For example, one simple implementation is to subtract images of the arm obtained by moving only its gripper, and to then perform trivial statistics on the thresholded image. This obtains the *perceived coordinates* of the hand: the coordinates of the hand image.

The computer which receives these values can also *send* signals to the arm. These consist of a number of values, one sent to each joint. Each specifies, in *encoder units*, the angle (or, for a prismatic joint, the length) that the joint should take. When these signals arrive the joint angles are automatically and slowly adjusted by independent servomechanisms until the specified values are all achieved. Thus, for hand-eye coordination, the **Action** is the set of requested joint angles.

A typical hand-eye coordination task is to move the perceived hand position to a target perceived position. This can be specified directly by showing the goal to the controller. It is clear that to achieve the task the controller needs knowledge of the relationship between the perceived image coordinates and the raw action signal it sends. It is interesting to note that it *does not* necessarily need to know relationships between these and any absolute "real world" coordinate system.

This is an example of the general problem where a controller needs a relationship between the raw actions it is meant to supply and the perceived behaviour. In the cases where there is domain

Bibliography

[Aboaf *et al.*, 1989] E. W. Aboaf, S. M. Drucker, and C. G. Atkeson. Task-Level Robot Learning: Juggling a Tennis Ball More Accurately. In *IEEE International Conference on Robotics and Automation*, 1989.

[Aha *et al.*, 1990] D. W. Aha, D. Kibler, and M. K. Albert. Instance-Based Learning Algorithms. *To appear in Machine Learning*, 1990.

[Albus, 1975a] J. S. Albus. A New Approach to Manipulator Control: Cerebellar Model Articulation Controller (CMAC). *Journal of Dynamic Systems, Measurement and Control*, pages 220–227, September 1975.

[Albus, 1975b] J. S. Albus. Data Storage in the Cerebellar Model Articulation Controller (CMAC). *Journal of Dynamic Systems, Measurement and Control*, pages 228–233, September 1975.

[Albus, 1981] J. S. Albus. *Brains, Behaviour and Robotics.* BYTE Books, McGraw-Hill, 1981.

[An *et al.*, 1988] C. H. An, C. G. Atkeson, and J. M. Hollerbach. *Model-Based Control of a Robot Manipulator.* M. I. T. Press, 1988.

[Angus, 1989] J. E. Angus. On the connection between neural network learning and multivariate non-linear least squares estimation. *Neural Networks*, 1(1), January 1989.

[Atkeson and Reinkensmeyer, 1989] C. G. Atkeson and D. J. Reinkensmeyer. Using Associative Content-Addressable Memories to Control Robots. A. I. Memo 1124, M. I. T. Artificial Intelligence Laboratory, November 1989.

[Atkeson, 1989] C. G. Atkeson. Using Local Models to Control Movement. In *Proceedings of Neural Information Processing Systems Conference*, November 1989.

[Barto *et al.*, 1983] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike Adaptive elements that that can learn difficult Control Problems. *IEEE Transactions on Systems Man and Cybernetics*, 1983.

[Bentley, 1980] J. L. Bentley. Multidimensional Divide and Conquer. *Communications of the ACM*, 23(4):214—229, 1980.

[Breiman *et al.*, 1984] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, 1984.

[Burghes and Graham, 1980] D. Burghes and A. Graham. *Introduction to Control Theory including Optimal Control.* Ellis Horwood, 1980.

[Burkhill, 1978] J. C. Burkhill. *A First Course in Mathematical Analysis.* Cambridge University Press, 1978.

[Christiansen *et al.*, 1990] A. D. Christiansen, M. T. Mason, and T. M. Mitchell. Learning Reliable Manipulation Strategies without Initial Physical Models. In *IEEE Conference on Robotics and Automation*, pages 1224–1230, 1990.

[Cleveland and Delvin, 1988] W. S. Cleveland and S. J. Delvin. Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting. *Journal of the American Statistical Association*, 83(403):596–610, September 1988.

[Cleveland, 1979] W. S. Cleveland. Robust Locally Weighted Regression and Smoothing Scatterplots. *Journal of the American Statistical Association*, 74(368):829–836, December 1979.

[Clocksin and Moore, 1989] W. F. Clocksin and A. W. Moore. Some Experiments in Adaptive State Space Robotics. In *Proceedings of the 7th AISB Conference, Brighton.* Morgan Kaufman, April 1989.

[Connell and Utgoff, 1987] M. E. Connell and P. E. Utgoff. Learning to control a dynamic physical system. In *Proceedings of the American Association for Artificial Intelligence Conference*, 1987.

[Diederich, 1990] J. Diederich. An Explanation Component for a Connectionist Inference System. In L. C. Aiello, editor, *Proceedings of ECAI90: the 9th European Conference on Artificial Intelligence*, pages 222–227, August 1990.

[Duda and Hart, 1973] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, 1973.

[Farmer and Sidorowich, 1988] J. D. Farmer and J. J. Sidorowich. Predicting Chaotic Dynamics. In J. A. S. Kelso, A. J. Mandell, and M. F. Shlesinger, editors, *Dynamic Patterns in Complex Systems.* World Scientific, New Jersey, 1988.

[Firschein and others, 1986] O. Firschein et al. Teleoperation and Robotics Scenarios. In *Artificial Intelligence for Space Station Automation*, pages 288–305. NASA Advanced Technology Advisory Committee, 1986.

[Franke, 1982] R. Franke. Scattered Data Interpolation: Tests of Some Methods. *Mathematics of Computation*, 38(157), January 1982.

[Friedman *et al.*, 1977] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Transactions on Mathematical Software*, 3(3):209–226, September 1977.

[Fu *et al.*, 1987] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee. *Robotics: Control, Sensing, Vision and Intelligence.* McGraw-Hill, 1987.

[Fu, 1970] K. S. Fu. Learning Control Systems—Review and Outlook. *IEEE Transactions on Automatic Control*, pages 210–221, April 1970.

[Gordon and Grefenstette, 1990] D. F. Gordon and J. J. Grefenstette. Explanations of Empirically Derived Reactive Plans. In *Proceedings of the 7th International Conference on Machine Learning*, June 1990.

[Gottschalk *et al.*, 1989] P. G. Gottschalk, J. L. Turney, and T. N. Mudge. Efficient Recognition of Partially Visible Objects Using a Logarithmic Complexity Matching Technique. *International Journal of Robotics Research*, 8(6), December 1989.

[Grosse, 1989] E. Grosse. LOESS: Multivariate Smoothing by Moving Least Squares. In *Approximation Theory VI.* Academic Press, 1989.

[Hearn, 1973] A. C. Hearn. REDUCE 2 User's Manual. Technical report, University of Utah, March 1973.

[Hoel, 1971] P. G. Hoel. *Introduction to Mathematical Statistics.* Wiley International, 1971.

[Holland *et al.*, 1987] J. H. Holland, L. B. Booker, and D. E. Goldberg. Classifier Systems and Genetic Algorithms. Technical Report No. 8, University of Michigan, Cognitive Science and Machine Intelligence Laboratory, 1987.

[Jordan and Jacobs, 1990] M. I. Jordan and R. A. Jacobs. Learning to Control an Unstable system with Forward Modeling. Technical report, M. I. T., 1990.

[Kaelbling, 1990a] L. P. Kaelbling. Learning Functions in k-DNF from Reinforcement. In *Proceedings of the 7th International Conference on Machine Learning*, June 1990.

[Kaelbling, 1990b] L. P. Kaelbling. Learning in Embedded Systems. PhD. Thesis. Technical Report No. TR-90-04, Stanford University, Department of Computer Science, 1990.

[Kibler *et al.*, 1988] D. Kibler, D. W. Aha, and M. K. Albert. Instance-Based Prediction of Real-Valued Attributes. Technical report 88-07, University of California, Irvine, 1988.

[Langley *et al.*, 1983] P. Langley, G. L. Bradshaw, and H. A. Simon. Rediscovering Chemistry with the BACON System. In *Machine Learning: An Artificial Intelligence Approach.* Morgan Kaufmann, 1983.

[Maclaren, 1989] N. M. Maclaren. Some Empirical Algorithms for non-Euclidean Nearest Neighbour Searching. Document at Cambridge University Computer Laboratory, December 1989.

[Mel, 1988] B. W. Mel. MURPHY: A Robot that Learns by Doing. Technical Report UIUCDCS-R-88-1397, University of Illinois at Urbana-Champaign, January 1988.

[Mel, 1989] B. W. Mel. MURPHY: A Connectionist Approach to Vision-Based Robot Motion Planning. Technical Report CCSR-89-17A, University of Illinois at Urbana-Champaign, June 1989.

[Michie and Chambers, 1968] D. Michie and R. A. Chambers. BOXES: An Experiment in Adaptive Control. In *Machine Intelligence 2*. Oliver and Boyd, 1968.

[Michie, 1989] D. Michie. Personal Models of Rationality. *Journal of Statistical Planning and Inference*, 21, 1989. Also published as a Turing Institute Technical Report.

[Miller *et al.*, 1987] W. T. Miller, F. H. Glanz, and L. G. Kraft. Application of a General Learning Algorithm to the Control of Robotic Manipulators. *International Journal of Robotics Research*, 6(2), 1987.

[Miller, 1989] W. T. Miller. Real-Time Application of Neural Networks for Sensor-Based Control of Robots with Vision. *IEEE Transactions on systems, Man and Cybernetics*, 19(4):825–831, July 1989.

[Minsky and Papert, 1969] M. Minsky and S. Papert. *Perceptrons*. M. I. T. Press, 1969.

[Moore, 1990] A. W. Moore. Acquisition of Dynamic Control Knowledge for a Robotic Manipulator. In *Proceedings of the 7th International Conference on Machine Learning*, June 1990.

[Murray, 1972] W. Murray. *Numerical Methods for Unconstrained Optimization*. Academic Press, 1972.

[Omohundro, 1987] S. M. Omohundro. Efficient Algorithms with Neural Network Behaviour. Technical Report UIUDCS-R-87-1331, University of Illinois at Urbana-Champaign, April 1987.

[Phan *et al.*, 1990] M. Phan, J. N. Juang, and R. W. Longman. Recent Developments in Learning Control and System Identification for Robots and Structures. In *Dynamics of Space structures Conference, Cranfield Institute of Technology*, June 1990.

[Poggio and Girosi, 1989] T. Poggio and F. Girosi. Regularization Algorithms for Learning That Are Equivalent to Multilayer Networks. *Science*, 247:978–982, 1989.

[Preparata and Shamos, 1985] F. P. Preparata and M. Shamos. *Computational Geometry*. Springer-Verlag, 1985.

[Quinlan, 1983] J. R. Quinlan. Learning Efficient Classification Procedures and their Application to Chess End Games. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning—An Artificial Intelligence Approach (I)*. Tioga Publishing Company, Palo Alto, 1983.

[Raibert, 1978a] M. H. Raibert. A Model for Sensorimotor Control and Learning. *Biological Cybernetics*, 29:29–36, 1978.

[Raibert, 1978b] M. H. Raibert. Motor Control and Learning by the State Space Model. PhD. Thesis, M. I. T., 1978.

[Rendell, 1983] L. Rendell. A New Basis for State Space Learning Systems and a Successful Implementation. *Artificial Intelligence*, pages 369–392, 1983.

[Rumelhart and McClelland, 1984] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing—Explorations in the Microstructure of Cognition.* M. I. T. Press, 1984.

[Salzberg, 1988] S. Salzberg. Exemplar-based learning: theory and implementation. Technical report, Aiken Computation Laboratory, Harvard University, 1988.

[Sammut and Michie, 1989] C. Sammut and D. Michie. Controlling a 'Black Box' Simulation of a Spacecraft. Technical Report TIRM-89-039, Turing Institute, October 1989.

[Samuel, 1967] A. L. Samuel. Some Studies in Machine Learning using the Game of Checkers II—Recent Progress. Memo No. 52, Stanford Artificial Intelligence Project, June 1967.

[Simons *et al.*, 1982] J. Simons, H. Van Brussel, J. De Schutter, and J. Verhaert. A Self-Learning Automaton with Variable Resolution for High Precision Assembly by Industrial Robots. *IEEE Transactions on Automatic Control*, 27(5):1109–1113, October 1982.

[Stanfill and Waltz, 1986] C. Stanfill and D. Waltz. Towards Memory-Based Reasoning. *Communications of the ACM*, 29(12):1213–1228, December 1986.

[Sutton, 1990] R. S. Sutton. Integrated Architecture for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In *Proceedings of the 7th International Conference on Machine Learning*, June 1990.

[Utgoff, 1989] P. E. Utgoff. Incremental Induction of Decision Trees. *Machine Learning*, 4:161–186, 1989.

[Valiant, 1984] L. G. Valiant. A Theory of the Learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[Vogel, 1989] T. U. Vogel. PhD. Thesis Proposal. First Year Report at University of Cambridge Computer Laboratory, May 1989.

[Whitley, 1989] D. Whitley. Applying Genetic Algorithms to Neural Network Learning. In *Proceedings of the 7th AISB Conference, Brighton.* Morgan Kaufman, April 1989.

[Zrimec and Mowforth, 1990] T. Zrimec and P. Mowforth. Learning by an Autonomous Agent in the Pushing Domain. In *Machine Intelligence 12 (to appear shortly)*. Oliver and Boyd, 1990.